STOL Directives

ITOS Edition

$Date: 2006/10/03 19:01:55 $

STOL Directives

Each of the following sections contains a summary of the directive syntax which conforms to the following conventions:

- Words in all capitals are literal keywords, and should be provided as-is.
- All symbols except square brackets ('`[]`'), vertical bar ('`|`'), elipses ('`...`') and expansion ('`==>`') are literal.
- Anything in square brackets ('`[]`') is optional.
- The symbol '`|`' means "or", so '`THIS|THAT`' means you can supply one of the literal strings "this" or "that".
- Words in italics are formal parameters for which a further syntax summary and or explanation will be provided.
- Normal text is explanatory material; for example: In '`NAME = filename`', '`filename`' is neither literal or a formal parameter. It is explaining that a filename should be given.
- Elipses ('`...`') mean that you may supply more than one of the preceding elements.
- The symbol '`==>`' following a formal parameter means "is replaced by". It us used to show what syntax may replace a formal parameter.
- Syntactic elements must be provided in the order given unless otherwise noted.
- Commas must be provided only as and where given in the syntax.

Formal parameters often are accompanied by their own syntax summary, which may in turn contain further formal parameters. Indented lines under the definition of a formal parameter give additional information on that parameter, and take two forms:

- Lines beginning with a formal parameter (italicised word) followed by '`==>`' expand the meaning of that parameter.
- Other lines, often beginning with a literal keyword, provide Background PROC "test" (ID# 1234) options which may be given for the formal parameter in question. If no elipses follow the parameter in the syntax, then only one option may substitute for it; else more than one option may be given. When multiple options are given, order is not important unless otherwise noted.

STOL is not case sensitive. All directives, variables, and mnemonics can be given in either or mixed case. String values in STOL may be case sensitive.

ACQUIRE	Control telemetry acquisition.
ARCHIVE	Control transfer frame or packet archiving.
ASK	Request interactive input.
BREAK	Break out of a DO loop.
BLIND	Turn command verification on or off.
BYPASS	Bypass command frame acceptance checks.
CFGMON	Control the configuration monitor/equation processor.
CLEAR	Clear the command buffer.
CLOSE	Close a device used by READ or WRITE.
CMD	Send A Spacecraft Command.
CONTINUE	Start The Next Iteration Of A DO Loop.

DATE	Set (Or Query) The TCW's Idea Of GMT.
DISABLE	Disable Commanding.
DO	Begin A DO WHILE Or DO UNTIL Loop.
DUMPFILE	Specify Dumpfile Name.
ELSE	Begin The ELSE Clause In A Compound IF.
ELSEIF	Begin An ELSEIF Clause In A Compound IF.
ENABLE	Enable A Set Of Directives.
ENDDO	End A DO WHILE Or DO UNTIL Loop.
ENDIF	End A Compound IF.
ENDPROC	End A Proc File.
FREE	Destroy Local Or Global Variables.
GET	Get values from programmable devices.
GLOBAL	Create Global Variables.
GO	Resume A Waiting Proc.
GOTO	Jump To The Specified Line In A Proc.
GPIB	Control GPIB device.
HOTKEY	Report stol status.
IF	Simple Or Compound IF.
KILLPROC	Kill All Procs.
LET	Assign A Variable Or Mnemonic.
LIMITS	Control Limit Checking.
LOAD	(LOADPKT) Send Image Load File To Spacecraft.
LOCAL	Create Local Variables.
LOG	Control Event Message Logging.
MODE	Modify Command Mode Parameters.
OPEN	Open File Or Device For READ Or WRITE.
PAGE	Control Display Pages.
PKTDUMP	Control Packet Dumps.
PKTTIMEOUT	Control Packet Timeout.
PLAYBACK	Play back an archive.
PLOT	Control Plots.
PREVIEW	Control Directive Preview mode.
PROC	The First Directive In A Proc.
PURGE	Remove Pending Commands From The Sent-Queue.
QUIT	Terminate Stol.
RAW	Send A Spacecraft Command Via Raw CCSDS Telecommand Packet.
RAWTF	Send A Spacecraft Command Via Raw Transfer Frame.
RRAW	Send A Spacecraft Command Via Raw Packet w/o headers.
READ	Read From A File Or Serial Port.
REM	Echo A Comment To The Event Log.
RESET	Set Spacecraft's Next Expected Seq Number To 0.
RESYNC	Synch Ground And Spacecraft Frame Seq. Number.
RETRLIM	Set Max Number Of Auto Cmd Retransmissions.
RETRY	Retransmit All Pending Commands.
RETURN	Return From A Proc.
SCEVENT	Send a Spacecraft Event (Used only by Triana STOL).

SEND	Transmit The Command Buffer To The Spacecraft.
SEQPRT	Control Sequential Prints.
SET	Control programmable devices.
SETCOEF	Control Analog Conversion Coefficients.
SETGRND	Set Ground's Next Expected Frame Seq Number.
SETVR	Set Spacecraft's and Ground's Next Expected Frame Seq Number.
SHOVAL	Echo Expressions To The Event Log.
SIM	Control The Internal Simulator.
SNAP	Take Page Snapshots.
SPEED	Control Proc Execution Speed.
START	Start A Proc.
STOP	Abort An Image Load.
STRIPCHART	Control a stripchart.
SYSTEM	Issue a UNIX command.
TFDUMP	Control Transfer Frame Dumps.
TIMEON	Perform Timeon Calculations.
TIMEOUT	Set Time Interval For Command Acknowledgement.
UNLOCK	Unlock The Spacecraft's FARM.
VC	Specify the Default Virtual Channel For Commands.
VERIFY	Compare Image Files.
WAIT	Halt A Proc.
WRITE	Write To A File Or Serial Port.
ZERO	Clear ITOS counters

ACQUIRE

The *ACQUIRE* directive controls telemetry acquisition.

Syntax

`ACQUIRE` [`ATTACH | MOD`] [*station*] [*filter ...*]

`ACQUIRE` may be abbreviated `AC`.

Start unpacking data or modify the filtering on a telemetry stream this workstation is unpacking.

station ↦ is a telemetry source name (or alias).

filter ↦ [`ADD | DROP`] [`NOT`] [*vcspec ...*]

vcspec ↦ *vcid* [`ADD | DROP`] [`NOT`] [ALL | *appidlist*]

vcid ↦ `VC0 | VC1 | ... | VC7`

appid list ↦ *appid* [, *appid*] ...

`ACQUIRE` `DETACH` *station*

Stop unpacking data from the given *station*.

`ACQUIRE` OFF

Stop all data unpacking on this workstation.

`ACQUIRE` QUERY

List stations (data sources) from which this workstation is acquiring and unpacking data.

Discussion

The `acquire` directive controls the unpacking of telemetry values into the Current Value Table (CVT).

Entering `acquire` without `attach` or `detach` starts unpacking telemetry from the given *station*. All previously entered `acquire`s are automatically given an `acquire off`.

The `acquire attach` directive starts unpacking data from the given *station* without interferring with telemetry from other *station*s. `acquire detach` stops unpacking data from the given *station* (only).

The `acquire mod` directive changes the filter on unpacking data from the given *station*. See below for more information on filtering.

The *station* argument must be the name assigned to a telemetry source in the Source Description File. The same source may have several names, and one source may be assigned the name `default`. This `default` source will be used if no *station* is specified.

The *filter* argument tells the system what data to acquire from the given *station*. The filter basically is a list of virtual channel IDs, each optionally followed by a list of application IDs. This gives the list of telemetry packets you wish to unpack. A VCID alone or followed by the keyword `all` selects all APPIDs on that channel.

The filter keyword `not` changes the sense of the filter from positive (get me these packets) to negative (get me everything *except* these packets). You can use the `not` before the whole list of items, or before a list of APPIDs in a *vcspec*. Note that double negatives are possible, but frowned upon.

The filter keywords `add` and `drop` are used in conjunction with the `acquire` argument `mod` to change the filter on an existing acquisition. Their use is evident from their names: `add` adds the given list of APPIDS to the list already being unpacked for the given *station*; and `del` deletes the given list of APPIDS to the list.

Entering `acquire` by itself will start unpacking all packets on the virtual channel given by `gbl_tm_defvc` from the default *station*.

The `acquire off` directive stops all telemetry unpacking on the workstation where it's exectuted. To stop a single acquisition when more than one is running, use `acquire detach`. To perform a modification like stopping decommutation of a virtual channel, use `acquire mod` with a filter containing the keyword `drop`.

By convention, the `acquire sim` directive starts acquisition of data from the internal simulator, but a properly formatted '`sim`' entry in the '`ctrlsource.dat`' file is required to support this. Simulated values can be changed with the SIM CHG directive.

ARCHIVE

The *ARCHIVE* directive controls the processes that archive transfer frames and packets.

Syntax

`ARCHIVE` [*station*] [*option ...*] [*filter*]

`ARCHIVE` may be abbreviated `AR`.

Start archiving telemetry data to a file.

station ↦ identifies the telemetry source.

option ↦ `PKT | FRAME`
`ID =` id
`FILE =` filename
`POOLSIZE =` size
`WRAPPERS =` *wrappers*

wrappers ↦
"*wrapper1* [*wrapper2*] ..."

filter ↦ if the `PKT` option given, a *packet filter*; else, a *frame filter*

frame filter ↦
`[ADD | DROP] [NOT]` [ALL | *vcid ...*]

packet filter ↦
vcid `[ADD | DROP] [NOT]` [ALL | *appid_list*]

vcid ↦ `VC0 | VC1 | ... | VC7`

appid_list ↦
appid [, *appid*] ...

`ARCHIVE` QUERY

Reports what's currently being archived.

`ARCHIVE` `STOP` *option*

Stops archiving.

option ↦ `ALL`
`FILE =` filename
`ID =` id

Discussion

The `archive` directive controls the way ITOS stores telemetry data into *archive* files. Archives are composed of one or more like-sized data files and a header file, and may hold frames or packets. Archive data files have an upcounting integer suffix counting from 0, while the header files have an ".H" suffix. Archives are created in the directory given by *GBL_TM_ARCHDIR*.

Use the first form given above to open a new archive or modify the filter on an already-open archive. (More about filters below.) Specify the *station* argument to select the data source providing the data you wish to archive, when more than one source is in use. Valid station names are in the Source Configuration File, and spacecraft data is the default source. When the data source is from an archive playback use "playback" as the source. Entering simply `archive` with no arguments opens an archive for all spacecraft data.

Specify the `pkt` option to archive packets and the `frame` to archive frames. The default is `frame`.

For new archives, the archive software automatically will generate a filename from the current date and time; however, you can use the `file` option to specify a different filename.

Instead of giving a filename, you can use the *id* option to identify the archive. The ID assigned to an archive with this option can then be used to refer to it in subsequent `archive` directives (in the same ITOS session). Any string may be used as the archive ID.

The *poolsize* option is provided for controlling the size of the individual archive data files. The default anticipates storing a 64 kilobit stream for 15 minutes and is set to just under 7.4 megabytes.

The *wrappers* option is provided for packet archives to specify the ordered list of wrappers to be added to each packet in the archive. Currently *anno12* and *anno12aos* are the only annotation wrappers available. The wrapper list must be space seperated list of wrapper names within a set of double quotes.

The filter controls what data is stored in the archive. For frame archives, the filter consists of a list of virtual channel IDs; for packet archives, the filter consists of application IDs. See the explanation of the *filter* argument to the ACQUIRE directive for more information. Just remember not to give any APPID lists when specifying the filter for a frame archive.

Use the `archive query` directive to get a report on what archives currently are open and what their filtering is. (The filter report doesn't work terribly well for packet archive unless they're small.) The report will include the archive ID, if you assigned one, and the archive filename along with the filter.

Use the `archive stop` directive to close archives. `archive stop all` closes all archives for all *station*s. To close a particular archive, use the `id` or `file` option to specify which archive to close.

ASK

Ask the operator a question in a pop-up window.

Syntax

`ASK` *prompt*[, *variable*]

prompt A string constant or parenthesized expression; the character `\n` split the prompt into multiple lines.

variable A variable name. If no variable is specified, a variable called `ANSWER` is assumed. If `ANSWER` does not exist, global `ANSWER` will be created.

Discussion

The `ASK` directive creates a popup window and suspends the current proc until the popup is dismissed. The popup displays the `prompt` string, has space for the test conductor to enter a response, and has an `ok` and a `cancel` button. The popup is dismissed when either button is clicked. If `ok` is clicked, the response (a string) gets assigned to the *variable*; if `abort` is clicked, the *variable* is unchanged.

STOL will not accept the `ASK` directive unless a proc is running.

Interactive directives are allowed while the popup is displayed.

Occasionally it's convenient for procs to ask the user whether or not to continue, instead of asking for a value. Here's one way:

```
answer = 0
ask "click OK to continue, ABORT to abort"
if (isint(answer)) return
rem ; OK ... continuing ...
```

The TODATE function is useful when asking for a date:

```
local date
ask "Enter the date"
date = todate(answer)
```

Finally, the STRTOL function lets you ask for a binary, octal, or hex value:

```
local binary, octal, hex
ask "Enter binary value"
binary = strtol(answer,2)
ask "Enter octal value"
octal = strtol(answer,8)
ask "Enter hex value"
hex = strtol(answer,16)
```

BREAK

Break out of a `DO` loop.

Syntax

```
BREAK
```

Discussion

The `BREAK` directive can only be used inside a DO loop, and is described with the `DO` directive. See [DO], page 20.

BLIND

Turn CCSDS command verification on or off.

Syntax

```
BLIND     ON | OFF
```

Discussion

The `BLIND` directive turns CCSDS command verification on or off. When command verification is off, the CLCW is not examined to determine which commands have been received by the FARM. Commands sent in blind mode automatically are purged from the Sent Queue.

This directive may optionally have a preceding slash, as in `/BLIND ON`. It is allowed only when commanding is enabled (see [ENABLE], page 24).

See *GBL_FOPSTATE_0* for a method of determining whether or not blind commanding is in effect.

BYPASS

Turn BYPASS FLAG on or off in CCSDS command transfer frames.

Syntax

```
BYPASS    ON | OFF
```

Discussion

The `BYPASS` directive controls whether or not the TC Transfer Frame BYPASS FLAG is set on subsequent commands. When the BYPASS FLAG is set, the spacecraft FARM bypasses its normal Frame Acceptance Checks. (The FARM applies Frame Validation Checks regardless of the setting of the BYPASS FLAG).

A `BYPASS ON` directive causes the BYPASS FLAG to be set on subsequent commands; a `BYPASS OFF` directives causes the BYPASS FLAG to be clear on subsequent commands. In normal operations the BYPASS FLAG is clear.

The `BYPASS` directive is used in emergency situations to force in data-carrying frames without regard for their sequentiality or for the command window in effect. It can also be used whenever telemetry is not available in order to avoid command verification failure (i.e. commands cannot be verified when telemetry is not avaliable). See section "Spacecraft Commands" in *The ITOS Command Subsystem*.

This directive may optionally have a preceding slash, as in `/BLIND ON`. It is allowed only when commanding is enabled (see [ENABLE], page 24).

CFGMON

Start or Stop a configuration monitor.

Syntax

`CFGMON` *configuration*

`CFGMON` `CLEAR` [*configuration* | `ALL`]

configuration
identifies which configuration monitor definition to start or stop.

Discussion

CLEAR

Clear the command buffer.

Syntax

`CLEAR`

Discussion

The `CLEAR` directive is used to remove all commands currently in the command buffer.
This directive may optionally have a preceding slash, as in `/CLEAR`.
This directive is only allowed when commanding is enabled (see [ENABLE], page 24).
This directive is not allowed when a `LOAD` (see [LOAD], page 44) is in progress.

CLOSE

Close an open device unit.

Syntax

`CLOSE` (*unit*)

unit is the same that was specified in the `OPEN` directive (see [OPEN], page 49).

Discussion

The `CLOSE` directive closes a unit previously opened with the `OPEN` directive.

CMD

Send a spacecraft command.

Syntax

`CMD` *cmdmnemonic field=value, ...*

`/` *cmdmnemonic field=value, ...*

cmdmnemonic
: The name of a command mnemonic, as specified in the database.

field
: The name of a field for *cmdmnemonic*, as specified in the database.

value
: The value for *field*. Either an expression or the name of a discrete value for *field*, as specified in the database.

"field=value"
: may be shortened to "*value*" when *value* is a discrete value and only one of *cmdmnemonic*'s fields uses that discrete value.

Discussion

The `CMD` directive sends a spacecraft command. `CMD` may optionally have a preceding slash, as in `/CMD SNOOP`.

This directive is only allowed when commanding is enabled (see [ENABLE], page 24).

Each command has zero or more fields. A value for each field may be specified (not more than once) in the `CMD` directive. The value for a field may be specified as an expression or by naming one of the field's discrete values. If a field has a discrete value named "DEFAULT", that field is optional and does not have to be specified in the `CMD` directive.

The field's limits and global mnemonic *gbl_rangechk* determine whether or not the field's value can be specified via expression. If *gbl_rangechk* is 0 (useful, for example, during I&T), the value may be specified via expression. Otherwise, if the field has at least one limit and the value is within the field's limits, the value may be specified via expression.

`CMD` examples:

```
/ANOP
/AECLIPSE ON
/MMMLOAD ADDRESS=50, NUMBYTES=2, VALUES=1
/MMMTABSEL tabid=1, selectram, dump
```

If the command is critical or hazardous then a POPUP window will open and require the operator to authorize the release of the command. Hazardous commands require a pass phrase to also be entered in the POPUP window that is compared against the global mnemonic variable *GBL_STOLHAZ_PHRASE*.

CONTINUE

Goto the ENDDO of a DO loop.

Syntax

```
CONTINUE
```

Discussion

The `CONTINUE` directive can only be used inside a DO loop, and is described with the `DO` directive (See [DO], page 20) and is only valid in `STOL` procs.

DATE

Set or query the ITOS GMT offset.

Syntax

`DATE`

`DATE` *date*

`DATE` `SYNC` [, *year*]

date A date constant or expression.

year An int constant or expression. Must range from 39 to 99(1900's), 2 to 38(2000's), or 1902 to 2038.

Discussion

The `DATE` directive is used to set the TCW's idea of Greenwich Mean Time. This is done by adjusting global mnemonic *gbl_gmtoff* which contains the offset in seconds from the TCW's internal clock and the simulated GMT.

There are three forms of the `DATE` directive:

`DATE` Query the current GMT:

```
DATE ⇒ Current date: 93-200-15:09:45:013000
```

`DATE` *date* Set the TCW's idea of GMT to the specified time. To advance the TCW's notion of GMT by one hour use `DATE P@GBL_GMTOFF + 60*60`. `DATE 94-124-16:24:38` is an example of syncing to a specific date/time.

`DATE SYNC [,` *year*`]`
Synchronize the TCW with the the date source. If no year is specified, the current year is assumed. The date source is identified by *gbl_date_source*, *gbl_date_txport*, and *gbl_date_type*.

DISABLE

The *Disable* an ITOS subsystem or query status.

Syntax

`DISABLE` *subsystem* [`STATUS`=*variable*]

`subsystem`
: Identifies the subsystem. May be one of `CMD`, `DSP`, `GPIB`, or `TLM`.

`STATUS`=*variable*
: Specifies a local or global variable or integer mnemonic to receive the status from the directive. States are 1 == SUCCESS, 0 == no change, -1 == ERROR.

Disable an ITOS subsystem. Each of these directives enables a suite of STOL directives and terminates certain processes related to the indicated subsystem.

- CMD - Disable the spacecraft command directives and terminates the `fopmux`, `fop`, and `cmd_transmit` programs. If the command simulator is running it will be terminated as well.
- TLM - Disable the telemetry directives and terminate the telemetry controller(`tmController`) and the `tlmClcw` and `tlmStatic` programs.
- DSP - Disable the display control directives, terminate the base pages and the data pointer server(`dp_server`) and Java display server(`IJServer`) programs.
- GPIB - Disable the GPIB/RS232 control directives and terminate the `deviceDriver` program if running.

`DISABLE` `QUERY` [`STATUS`=*variable*]

`STATUS`=*variable*
: Specifies a local or global variable or integer mnemonic to receive the status from the directive. The value returned is an integer with a bit weight as follows to identify if a subsystem is disabled:

  '8' CMD is enabled.

  '4' DSP is enabled.

  '2' GPIB is enabled.

  '1' TLM is enabled.

Print to the event log the list of subsystems and whether each is enabled or disabled. If a `STATUS` variable is option is used, A integer value is returned as in the example below:

```
global x
disable query status=x
if (BWAND(x, 1) .EQ. 0) enable tlm
```

In the above example, if x is returned the value of 4, this indicates the DSP system is enabled and everthing else is disabled. This would cause the `IF` to evaluate true and the "enable tlm" directive would be issued.

If `STATUS` variable is not used then a status message for each subsystem will print in the event log as below:

```
STOL_MSG: The cmd subsystem is DISABLED
STOL_MSG: The display subsystem is ENABLED
STOL_MSG: The GPIB subsystem is DISABLED
STOL_MSG: The telemetry subsystem is ENABLED
```

Discussion

Disables the specified subsystem. See [ENABLE], page 24 for opposite directive.

DO

The DO (and related) directives control looping in stol procs.

Syntax

DO WHILE (*condition*)

DO UNTIL (*condition*)

condition is any integral expression.

Discussion

DO WHILE A DO WHILE loop repeats as long as its condition evaluates non-0. A DO WHILE loop tests its condition at the top of the loop; if the condition is initially 0 the body of the loop is skipped.

DO UNTIL A DO UNTIL loop repeats as long as its condition evaluates to 0. A DO UNTIL loop tests its condition at the bottom of the loop; the body of the loop gets executed at least once regardless of the initial value of the condition.

BREAK The BREAK directive breaks out of the innermost loop, and is equivalent to a GOTO the directive following the loop's ENDDO.

CONTINUE The CONTINUE directive begins the next iteration of the innermost loop, and is equivalent to GOTO the loop's ENDDO.

ENDDO The ENDDO is the termination or end point of a DO loop. It designates the boundry point of the scope of the loop. When the loop terminates, execution begins at the next directive following the ENDDO.

Examples of common usage are:

```
DO WHILE ( condition )
   .
   BREAK
   .
   .
ENDDO

DO UNTIL ( condition )
   .
   .
   CONTINUE
   .
ENDDO
```

DUMPFILE

Sets the dump file basename.

```
DUMPFILE filename
```

Sets *gbl_dumpname* to *filename*. If *filename* is omitted, '`LOADFILE`' is the used.

See *gbl_dumpname*, *gbl_imgdumpdir*, and section "Image Dumps" in *The ITOS Command Subsystem*.

> This directive is only allowed when commanding is enabled (see [ENABLE], page 24).

ELSE

The `ELSE` directive may only be used inside a compound `IF` directive, and is described as part of the `IF` directive. See [IF], page 39.

ELSEIF

The `ELSEIF` directive may only be used inside a compound `IF` directive, and is described as part of the `IF` directive. See [IF], page 39.

ENABLE

The *ENABLE* directive enables the specified ITOS subsystem.

Syntax

`ENABLE` *subsystem* [`STATUS`=*variable*]

`subsystem`
: Identifies the subsystem. May be one of `CMD`, `DSP`, `GPIB`, or `TLM`.

`STATUS`=*variable*
: Specifies a local or global variable or integer mnemonic to receive the status from the directive. States are 1 == SUCCESS, 0 == no change, -1 == ERROR.

Enable an ITOS subsystem. Each of these directives enables a suite of STOL directives and starts certain processes related to the indicated subsystem.

- CMD - Enable the spacecraft command directives and start the `fopmux`, `fop`, and `cmd_transmit` programs.
- CMD, SIM - Enable the command subsystem with the command simulator.
- TLM - Enable the telemetry directives and start the telemetry controller(`tmController`) and the `tlmClcw` and `tlmStatic` programs.
- DSP - Enable the display control directives, start the base pages and the data pointer server(`dp_server`) and Java display server(`IJServer`) programs.
- GPIB - Enable the GPIB/RS232 control directives and start the `deviceDriver` program.

`ENABLE` `QUERY` [`STATUS`=*variable*]

`STATUS`=*variable*
: Specifies a local or global variable or integer mnemonic to receive the status from the directive. The value returned is an integer with a bit weight as follows to identify if a subsystem is enabled:

  '8' CMD is enabled.

  '4' DSP is enabled.

  '2' GPIB is enabled.

  '1' TLM is enabled.

Print to the event log the list of subsystems and whether each is enabled or disabled. If a `STATUS` variable is option is used, A integer value is returned as in the example below:

```
global x
enable query status=x
if (BWAND(x, 1) .EQ. 0) enable tlm
```

In the above example, if x is returned the value of 4, this indicates the DSP system is enabled and everthing else is disabled. This would cause the `IF` to evaluate true and the "enable tlm" directive would be issued.

If `STATUS` variable is not used then a status message for each subsystem will print in the event log as below:

```
STOL_MSG: The cmd subsystem is DISABLED
STOL_MSG: The display subsystem is ENABLED
STOL_MSG: The GPIB subsystem is DISABLED
STOL_MSG: The telemetry subsystem is ENABLED
```

Discussion

A long, long time ago, before ITOS was called ITOS, we divided it into six subsystems: command, telemetry, display, stol, database, and device control (gpib). All of the STOL directives fall into one of these catagories.

Originally, the `enable` and `disable` directives controlled only the command subsystem, since it is enabled only on one workstation in a cluster. More recently it became clear to us that we needed to be able to disable and enable other subsystems, if for no other reason than that we needed to reset them from time to time.

The strictly STOL directives – proceedure control, expression evaluation, and variable assignment – cannot be disabled or enabled. The database-related directives, such as `limit`, also cannot be disabled or enabled.

[We could go into considerably more detail on what happens when each subsystem is enabled.]

See [DISABLE], page 18 for opposite directive.

ENDDO

The `ENDDO` directive may only be used to end a `DO` loop, and is described as part of the `DO` directive. See [DO], page 20.

ENDIF

The `ENDIF` directive may only be used to end a compound `IF` directive, and is described as part of the `IF` directive. See [IF], page 39.

ENDPROC

The `ENDPROC` directive may only be used to end a proc, and is described as part of the `PROC` directive. See [PROC], page 63.

FREE

```
FREE name [, name ...]
```

name – a variable name.

The FREE directive directive is used to destroy local or global variables. In

```
GLOBAL one, two
LOCAL two, three, four
FREE one, two, three
```

the FREE directive destroys global variable (see [GLOBAL], page 33) one and local variables (see [LOCAL], page 45) two and three. After the FREE directive, global variable two and local variable four remain.

(Global variable two remains because the FREE destroyed local variable two. Between the LOCAL and FREE directives two different variables, local two and global two, existed (however, only local two could be seen or referenced)).

GET

```
GET device pid [= par] [variable]
```

device – A device is a string that matches a device name in the `gbl_dev_type` array such as (`BITSYNC`, `PSK`, `RECORDER`, `RECEIVER`, etc.).

pid – Parameter identification. The name of the particular function being queried.

=*par* – Optional parameter value for *pid*.

variable – Optional `LOCAL` variable, `GLOBAL` variable, or mnemonic to put returned value with any appropriate conversion done. If no name is given the returned value is written to *gbl_gpib_rcvmsg* as an ASCII character string.

Examples:

```
GET BITSYNC ID

GET BITSYNC ID MY_MNEMONIC

GET TEKSCOPE CHANNEL=1
```

The GET directive is used to query values from various devices capable of being controlled remotely over a GPIB or RS-232 interface. The definition of devices and their commands are in the file Device.conf. This directive relies on the settings of global mnemonics to determine which model of a particular device is being used and the communications setup parameters required for the device. For example, the Device.conf configuration file may define the commands for two different external devices, a bit synchronizer and an oscilliscope (eg. Loral DBS 430, and TEKTRONICS 2430). The global mnemonics for the devices, found in *ITOS_GPIB*, specify which device is being used such as:

```
GBL_DEV_TYPE[0]  = BITSYNC
GBL_DEV_MODEL[0] = LORALDBS430
GBL_DEV_IF[0]    = RS232
GBL_DEV_PORT[0]  = /dev/ttya
GBL_DEV_BAUD[0]  = 9600
GBL_DEV_CLEN[0]  = 8
GBL_DEV_SBIT[0]  = 1

GBL_DEV_TYPE[1]  = TEKSCOPE
GBL_DEV_MODEL[1] = TEK2430
GBL_DEV_IF[1]    = GPIB
GBL_DEV_PORT[1]  = GPIB-ITOS
GBL_DEV_ADDR[1]  = 4
```

A default Device.conf file comes with ITOS. The user can modify this file or create their own Device.conf and set the global *gbl_devcfgdir* to the directory where it is located. This is an example of the entries that need to be added to the Device.conf file in order to use the GET and SET directives.

```
$PIDS                 # DO NOT ERASE THIS LINE
#
# PID         FIXED/VARIABLE     Comment
#-------------------------------------------------
ID,           FIXED              # ID of Device
CHANNEL1,     FIXED              # Info of Channel 1
CHANNEL,      VARIABLE           # Info on a Channel
BR,           VARIABLE           # Bit Rate
$END                  # DO NOT ERASE THIS LINE - ADD CMDS BEFORE THIS LINE

$MODEL=TEK2430
#
# Digital Oscilloscope, Model Tektronix 2430A
#
# PID:PAR Pair ASCII Cmd String,   Comment
#----------------------------------------
ID,           "ID?"               # Identify device
CHANNEL1,     "CH1?"              # Channel 1 information
CHANNEL,      "CH%9%?"            # Channel 1 information
BR,           "B%9v999\.E9%."     # Bit rate
$END
```

For VARIABLE parameters such as bit rate(BR) shown above a COBOL-style PIC format enclosed within %'s is used and is described below.

The PIC may contain the following characters:

'+', '-'	:	The formatted number will begin with a '+' or '-'.
'9'	:	Used to specify field width. Each '9' is used to represent a digit.
'.'	:	Used for floating point numbers to specify decimal point placement.
'E','e'	:	Specifies the number to be formatted using exponentiation using the letter 'E'.
'Z','z'	:	Whenever a 'Z' appears before the digit specifier, leading zeros will be suppressed. By default, the field width is padded with zeros.
'V','v'	:	Used to show placement of assumed decimal point.
'\'	:	Precedes literal characters in the formatted number string. At this time literals may only be placed immediately preceding the

```
                    exponent character 'E'.

PIC Examples:

    NUMBER        PIC                FORMATTED NUMBER
    ------------------------------------------------------

    12345         +999999.9          +012345.0
    12345         Z999999.Z9         12345.
    12345         9999               2345
    12345.67      Z9.999999E+999     1.234567E+004
    12345         9.999EZ99          1.234E4
    123.45        9.9999E99          1.2345E02
    123.45        9.9E9              1.2E2
    123456        9v999\:E9          1234:E5
```

Also see [SET], page 80, [ENABLE], page 24 and [DISABLE], page 18.

GLOBAL

```
GLOBAL name [, name ...]
```

name – a variable name.

The GLOBAL directive creates global variables. There is no problem with (and no error event messages are produced if) the same name is used in more than one GLOBAL directive, for example:

```
GLOBAL summer, spring, winter, fall
GLOBAL winter
```

Note that if the same name is used in both LOCAL (see [LOCAL], page 45) and GLOBAL directives, two different variables will be created but only the LOCAL variable will be visible. However, trying to create a GLOBAL variable where a LOCAL variable already exists will cause a STOL error to be generated and stop the running proc.

A GLOBAL variable defined more than once will be ignored and cause a STOL warning message. A running proc will continue to run.

GLOBAL variables may also be destroyed using the FREE (see [FREE], page 29) directive.

GO

```
GO
```

The `GO` directive restarts a waiting proc. The `GO` directive should only be entered interactively since it makes no sense to have a `GO` directive in a proc file.

See [WAIT], page 105.

GOTO

```
GOTO linenumber
 - or -
GOTO label
```

linenumber – (an integer constant or expression) is the line number to goto.

label – (a string constant or expression) is the label to goto.

The `GOTO` directive jumps to the specified line or label (directive labels) in the current proc.

`GOTO` must not be used to jump in or out of `IF`node if or `DO` blocks; it this happens, mysterious error messages will result!

GPIB

```
GPIB POLL host, address

GPIB SEND host, address, message

GPIB RECEIVE host, address, msglen [, variable]
```

host – string expression containing the host name or IP address of the GPIB-Ethernet interface box. If the string contains any special characters is must be quoted.

address – integer expression from 0 thru 0x7E30. The primary address must be an integer from hex 00 thru hex 1E (0 to 30 decimal). If secondary addressing is used, the primary address appears in the low order byte and the secondary address appears in the high order byte. The secondary address must be an integer from hex 60 to hex 7E (96 to 126 decimal). For example, the address 0x6105 indicates a device whose primary address is 5 and whose secondary address is 0x61. NOTE: Some devices assume a secondary address offset of hex 60, so a displayed secondary address of 1 is really hex 61.

message – string expression or constant.

msglen – integer expression specifying the maximum length of the message to be received.

variable – Optional `LOCAL` variable, `GLOBAL` variable, or mnemonic to put `RECEIVE` value.

`GPIB POLL` The GPIB device is sent a serial poll for status. The status value returned by the device is a one byte integer that is put in global mnemonic *gbl_gpibserpoll* if it exists otherwise an event message will display the status value.

`GPIB SEND` The GPIB device is sent the command in *message*. If a command generates a response then you must issue a GPIB RECEIVE to retrieve it.

`GPIB RECEIVE`
The value received from the device is written to the optionally specified STOL variable or mnemonic with any appropriate conversion done. If none is specified then the returned value is written to global mnemonic *gbl_gpib_rcvmsg* as an ASCII character string.

Examples:

```
GPIB POLL gpib3, 7
STOL_MSG: sending message to device...
STOL_MSG: Received GPIB serial poll status byte of: 0

GPIB SEND "gpib-itos", 4, "ID?"
STOL_MSG: sending ID? comand to device at address 4 through host gpib-itos
```

```
GPIB RECEIVE "gpib-itos", 4, 100
```
STOL_MSG: Received GPIB msg and set GBL_GPIB_RCVMSG to: xxxxxxxxxxxxxxxx

```
GPIB RECEIVE "gpib-itos", 4, 12, gbl_char
```
STOL_MSG: Received GPIB msg and set GBL_CHAR to: ID TEK/2430A

HOTKEY

```
HOTKEY

HOTKEY > destination

HOTKEY >> destination
```

destination – A relative or absolute file name. Names the file the report gets written to (`>` means to overwrite that file, `>>` means to append to that file). If no destination is specified, the report gets written to the event log.

The `HOTKEY` directive generates a report showing current stol status. This directive is called `HOTKEY` because it is intended to be invoked via a hotkey.

IF

```
IF ( condition ) directive

IF ( condition ) THEN
   .
   .
ELSEIF ( condition ) THEN
   .
   .
ELSE
   .
   .
ENDIF
```

condition – an integer expression.

directive – any directive other than `DO`, `ELSE`, `ELSEIF`, `ENDIF`, `ENDPROC`, `IF`, or `PROC`.

This directive may only be used in procs.

The `ELSEIF` and `ELSE` clauses are optional. Multiple `ELSEIF` clauses are allowed.

The `IF` directive allows conditional execution of other directives.

`IF ( condition ) directive`
: The one-line IF executes <directive> only if <condition> .NE. 0. The one-line IF may be used interactively as well as inside procs.

`IF ( condition ) THEN`
: Begins a compound IF. The compound IF executes at most one of its clauses – the first whose condition .NE. 0 or, if all conditions .EQ. 0, the ELSE clause.

`ELSEIF ( condition ) THEN`
: Begins an ELSEIF clause in a compound IF. ELSEIF clauses are optional; multiple ELSEIF clauses may be specified.

`ELSE`
: Begins the ELSE clause in a compound IF. The ELSE clause is optional.

`ENDIF`
: Terminates a compound IF.

IF statements can be nested up to 40 levels deep for example:

```
IF (x > 10) THEN
  IF (x > 9) THEN
    IF (x > 8) THEN
      .
      .
    ENDIF
  ENDIF
ENDIF
```

KILLPROC

```
KILLPROC

KILLPROC BACKGROUND pid
```

KILLPROC may be abbreviated KP.

BACKGROUND may be abbreviated BG.

pid – expression representing the process number of a running background proc. *gbl_system_pid* can be used to for the last background proc started.

The KILLPROC directive with no arguments terminates all foreground procs.

The KILLPROC BACKGROUND directive terminates a particular background proc.

Use the RETURN (see [RETURN], page 76) directive to terminate the current proc only.

For example, the following proc fragment

```
GLOBAL pid

start test &
==>STOL_MSG: Background PROC "test" (ID# 1234) started.
pid = gbl_system_pid
==>STOL_MSG: Opened test in /home/itos/procs/test.proc

kp bg pid
==>STOL_MSG: Killing Background Task "[proc]test" (ID# 1234)
==>STOL_MSG: Background Task "[proc]test" (ID# 1234) exited.
```

See [START], page 90 for how to start a proc.

LET

```
LET name = value
 - or -
name = value
```

name – a variable or mnemonic name.

value – a constant or expression representing the value to be assigned to `name`.

The `LET` directive assigns a value to a local variable, global variable, or telemetry mnemonic.

An event message is generated that shows the value that was assigned. This is particularly useful if `value` is an expression or had to undergo a conversion before it could be assigned.

LIMITS

```
LIMITS ON list-of-mnemonics

LIMITS OFF list-of-mnemonics

LIMITS QUERY list-of-mnemonics

LIMITS DELETE list-of-mnemonics

LIMITS CHANGE mnemonic rl1, yl1, yh1, rh1, rl2, yl2, yh2, rh2
```

list-of-mnemonics – comma-seperated list of telemetry mnemonic names or the keyword `ALL`. `list-of-mnemonics` is optional for `LIMITS ON` and `LIMITS OFF` (`ALL` is assumed); `ALL` is not allowed for `LIMITS QUERY`.

mnemonic – a telemetry mnemonic name.

rl1 ... rh2 – floating point constant or expression or the keyword `NOCHG` (or nothing at all) representing the new red low 1, yellow low 1, ..., red high 2 limit values. A `NAN` or `INF` turns off limit checking for a particular red or yellow limit.

`LIMITS` may be abbreviated `LIMIT` or `LIM`.

`CHANGE` may be abbreviated `CHG`.

`DELETE` may be abbreviated `DEL`.

`LIMITS ON`
`LIMITS OFF`

Turns limit checking on or off for the specified mnemonics. If no mnemonics are specified, turns limit checking on or off for all mnemonics.

`LIMITS QUERY`

Reports the current limit settings for the specified mnemonics. The report is displayed in the event log as `STOL_MSG` events and has the form:

```
mnemonic: on_off rl1, yl1, yh1, rh1
 - or -
mnemonic: on_off rl1, yl1, yh1, rh1, rl2, yl2, yh2, rh2
```

`LIMITS DELETE`

Removes the limits settings for the specified mnemonics from the limits table. If the limit is shared it only removes the reference to those mnemonics not the limit entry.

`LIMITS CHANGE`

Change/create the specified limit values for the specified mnemonic. If limits don't currently exist they will be created. For example:

```
LIMITS CHANGE ACT2SC1 nan,,4.2
```

changes `ACT2SC1`'s red low 1 value off and yellow high 1 limit to `4.2`. *Warning:* Changing limits for a mnemonic that is shared by other mnemonics effectively changes it for all mnemonics sharing that limit set.

LOAD

```
LOAD filename

LOADPKT filename
```

filename – A string constant or parenthesized expression evaluating to a string naming a load file.

These directives may optionally have a preceding slash, as in `/LOAD initfile1`.

These directives are only allowed when commanding is enabled (see [ENABLE], page 24).

The LOAD directive is used to uplink a raw image to the spacecraft. *filename* names a raw image load file: first a formatted image load file is generated; then the formatted image load file is uplinked. The formatted image load file's name is *filename* with the extension changed to '`.PKT`'. For example, if *filename* is '`test3.ats`' the generated file will be named '`test3.PKT`'. Generated files get placed in the same directory as the original file.

The LOADPKT directive is used to uplink a formatted image to the spacecraft. *filename* names the formatted image load file to be uplinked.

If in ONESTEP command mode (see [MODE], page 48), the Transfer Frames are immediately uplinked to the spacecraft. If in TWOSTEP command mode, the Transfer Frames are stored in the command buffer pending transmission until the `SEND` directive is issued (see [SEND], page 78).

`filename` is the name of the image load file. If `filename` contains a `/` it is the pathname to the file; otherwise `filename` names the file in the default load file directory (global mnemonic *gbl_imgloaddir*).

For the formats of load files, see section "Image Loads" in *The ITOS Command Subsystem*.

When global mnemonic *gbl_loaddone* equals `1`, the load has been transmitted and received by the spacecraft.

Global mnemonic *gbl_loadfile* contains the name of the most recently load image file.

Use the `STOP` directive (see [STOP], page 92) to abort a load in progress.

LOCAL

```
LOCAL name [, name ...]
```

name – a variable name

The LOCAL directive creates local variables, and is intended to be used in proc files (it rarely makes sense to issue a LOCAL directive interactively). It is possible for a local variable to have the same name as a global variable or telemetry mnemonic, in which case the global variable or telemetry mnemonic is hidden until the local variable is destroyed. Local variables are automatically destroyed when their proc file returns or is killed; local variables may also be destroyed using the FREE directive.

A LOCAL variable defined more than once will cause an error to be displayed by STOL and the running proc will stop pending operator interaction.

See [GLOBAL], page 33. See [FREE], page 29.

LOG

```
LOG > file [PUSH]

LOG >> file [PUSH]

LOG POP

LOG QUERY

LOG STOP - depricated
```

file – log file name. Typically a quoted string (quoted because it normally contains special characters); can also be a parenthesized expression.

> *file* – create or overwrite the log file.

>> *file* – create or append to the log file.

PUSH – optional parameter specifies that the current log file is pushed onto stack before new log file is open. This then can be reversed by a LOG POP directive. Without this option the new log file replaces the current open log on the top of the stack.

The LOG directive controls event message logging.

Logging is turned on automatically whenever ITOS starts using '**startup_log**' file in the **gbl_evtldir** directory. The problem is that the log file is hard coded to get replaced each time ITOS starts. Obviously, if the file gets replaced, the previous contents are lost. But if the file gets appended to the file can become so large that it becomes extremely difficult to search! The solution is to use a different log file each time ITOS starts a startup proc using a unique log file name. One way to do this is:

```
LOG >> (CONCAT("LOG.",SUBSTR(P@GBL_GMTOFF,1,6)))
```

This logs to a file named "LOG.93-344" (assuming the TCW thinks today is Dec 10 1993). This file gets created if it didn't already exist and gets appended to if it did previously exist. The current log file is is closed before the new one is opened.

A word of warning: log files take up lots of disk space, and the disk can become full unless log files are carefully managed.

LOG STOP - is depricated. This directive will act like a LOG POP closing the current log and reopening the previous. You can stop logging by directing the log to '**/dev/null**' which in effect throws data away. This would be done by

```
LOG > "/dev/null"
```

but this is not recommended.

LOG POP - reverses a previous LOG file PUSH directive by closing the current log file and reopening the previous log file for append. You can not pop off the lowest (or first) log file since a log file must always be open.

LOG QUERY - reports the current log file being written as well as any pushed on the stack.

See section "Logging events" in *Event Processing*

MODE

```
MODE STEP1
 - or -
MODE STEP2

MODE BYPASS ON
 - or -
MODE BYPASS OFF

MODE REXMIT ON
 - or -
MODE REXMIT OFF

MODE VERIFICATION ON
 - or -
MODE VERIFICATION OFF

MODE VTIME timeout
```

This directive may optionally have a preceding slash, as in `/MODE REXMIT ON`.

This directive is only allowed when commanding is enabled (see [ENABLE], page 24).

The `MODE` directive controls various command parameters.

`MODE STEP` Selects onestep or twostep commanding.

`MODE BYPASS`
: `MODE BYPASS ON` is equivalent to `BYPASS ON`, and `MODE BYPASS OFF` is equivalent to `BYPASS OFF`. See [BYPASS], page 11.

`MODE REXMIT`
: `MODE REXMIT ON` is equivalent to `RETRLIM 3`, and `MODE REXMIT OFF` is equivalent to `RETRLIM 0`. See [RETRLIM], page 74.

`MODE VERIFICATION`
: `MODE VERIFICATION ON` is equivalent to `BLIND OFF`; `MODE VERIFICATION OFF` is equivalent to `BLIND ON`. See [BLIND], page 10.

`MODE VTIME`
: `MODE VTIME timeout` is equivalent to `TIMEOUT timeout`. See [TIMEOUT], page 101.

OPEN

```
OPEN ( unit ) thing [opts ...]

OPEN ( unit ) QUERY [opts]

OPEN QUERY
```

unit – a number (an integer constant or an expression that evaluates to an integer) that identifes *thing* to subsequent READ, WRITE, or CLOSE directives.

thing – a string (or expression that evaluates to a string) that identifes the file, port, or socket to be opened. See the discussion below.

opts – a space seperated list of options. See the discussion below.

The *OPEN* directive opens a file, serial port, or socket for use by the READ and WRITE directives or requests status of one or all open units.

Opening Files

If *thing* identifies a FIFO or file, then *opts* can be:

`READ`
`WRITE`
`RDWRITE` Defaults to RDWRITE.

fileoptions ⇒ `CREATE`
fileoptions ⇒ `APPEND`
Defaults to APPEND.

`STATUS=`*variable*
Specifies a local or global variable or integer mnemonic to receive the status from the open operation.

States are 1 == SUCCESS, -1 == ERROR.

Using this option will cause a proc to continue regardless of error except a panic message.

It is up to the user to test the return *variable* before attempting to READ or WRITE.

Opening Serial Ports

If *thing* identifies a serial device, then *opts* can be:

`BAUD =` *baudrate*
baudrate must be a rate supported by the host hardware and operating system. Rates of 300, 1200, 2400, 4800, 9600, 19200, and 38400 are supported on nearly all systems. Defaults to 9600.

`CS7`
`CS8` Sets 7 or 8 bit characters. 7 bit characters have two stop bits; 8 bit characters have 1 stop bit. Defaults to 7 bit characters.

`PARITY =` *parity*
`EVEN`, `ODD`, or `NONE`. Defaults to `EVEN`.

`STATUS=`*variable*
same as Opening Files.

For example, to open the S0 serial port on Linux at 38400 baud with 8 bit character length, 1 stop bit, and no parity:

```
OPEN (1) "/dev/ttyS0" BAUD=38400 CS8 PARITY=NONE
```

Opening sockets

If *thing* contains a colon, it has the form "*host*:*port*" (or simply ":*port*") and indicates a socket, then *opts* can be:

`TCP`
`UDP` Type of socket connection. Defaults to `TCP`.

`SERVER` Only valid for sockets. The unit is opened as a socket server. If not specified, the socket is opened as a `CLIENT`. `SERVER` doesn't make sense with `UDP`.

`AUTORCN` Only valid for server sockets. The unit is opened for `AUTO RECONNECT`, that is if the connection is broken it will return to listen mode and accept a connection from a new `CLIENT`.

`STATUS=`*variable*
Specifies a local or global variable or integer mnemonic to receive the status from the open operation. A local variable is NOT allowed with the SERVER option since a proc could exit before the connect is complete making the status variable invalid.

States are 1 == SUCCESS, 0 == NO CONNECT, -1 == ERROR.

This option will cause a proc to continue on any status value except ERROR with a panic message.

It is up to the user to test the return *variable* before attempting to READ or WRITE. Only a status of SUCCESS actually has an open socket connection.

`TIMEOUT=`*seconds*
Sets the maximum number of seconds a STOL proc will wait for a socket connection to be made or accepted. If this option is not specified, a default timeout of `30.0` seconds is assumed.

A proc is suspended until a connection is made, an error is detected or `TIMEOUT` is exceeded. An interactive `GO` directive causes the `OPEN` to immediately time out. A timeout will close the unit and the proc will halt unless the `STATUS` option is specified. A `0.0 TIMEOUT` waits indefinitely.

`TIMEOUT` is ignored with `UDP` or `STATUS` options or on an interactive `OPEN`.

Open Unit Query

`OPEN UNIT QUERY` reports status of one unit. The *opt* can be:

`STATUS=`*variable*
: Specifies a local or global variable or integer mnemonic to receive the status from the open operation. States are 1 == OPEN, 0 == NOT CONNECTED, -1 == NOT OPEN.

  A STATUS of '1' indicates the unit is open. A STATUS of '0' indicates the unit is open but that a socket connection is not complete to the other end. A STATUS of '-1' indicates no such unit number is currently open.

Open Query

`OPEN QUERY` reports which things are currently open.

See Also

See [CLOSE], page 14 for how to close an opened unit.

See [READ], page 69 for how to read from an open unit.

See [WRITE], page 107 for how to write to an open unit.

PAGE

The *PAGE* directive controls the telemetry display pages.

Syntax

`PAGE` *name*`[,` *slot*`][,` *rate*`]`

Bring up a new page; bring a running page to the top, change its refresh rate, or move it to a new slot.

name String constant or expression naming a page.

slot Integer constant or expression giving a location on the screen where the page should be displayed.

rate Floating point constant or expression giving the number of seconds between page data updates.

`PAGE CLEAR` *name*
`PAGE CLEAR ALL`

Remove a page from the display, or remove all but base pages from the display.

`PAGE FREEZE` *name*

Stop a page from updating.

`PAGE THAW` *name*

Undo a freeze; allow a page to start updating again.

`PAGE REFRESH` *name*`[,` *rate*`]`

Change a page's update rate, or cause it to update immediately

Discussion

The `page` directive controls telemetry mnemonic and other page displays. Telemetry mnemonic pages are defined in disk files according to the rules set out in the ITOS Page & Seqprt Definition Guide. Telemetry pages can also be created with the `makepage` utility.

The page directive also controls the following set of *special* pages:

'`sentq`' Command sent queue display. This page displays the FOP sent queue, giving verification status for the last 128 commands sent.

'`cmdbuf`' Command buffer display. This page displays the command buffer, used in two-step commanding and for sending spacecraft loads.

'`control`' Display control page. This page lists all active pages, snaps, packet & frame dumps, sequential prints, plots, and configuration monitors.

'`pktcount`'

Telemetry packet counter page. This page shows how many of each packet in the database has been processed by the current instance of the unpacker program, `tlmClient`.

'`limitview`'

Display section "dsp_limitview" in *Limit Viewer* page. This page lists all of the mnemonics with current limit violations.

Use the first form above to start a page. The *name* argument can be:

1. The complete path of a page file, including its extension, enclosed in quotes.
2. The name of the page, not including the ".page" or ".disp" extension. The system will search the directories given by `gbl_pagepath` for the page definition file.
3. One of the special pages mentioned above. The system looks for a special page executable called '`dsp_`'*name* in the directory given by `gbl_dspbin`.

The *slot* argument directs where the page should be displayed. The screen may be thought of as containing four quadrants: Slot 1 is in the lower left, just below the main event window, slot 2 is in the lower right, also just below the event window, slot 3 is in the upper left, just below the STOL input window and above slot 1, and slot 4 is in the upper right, above slot 2.

When running under the `olvwm` window manager, you also can bring pages up on different virtual screens. In this case, the *slot* argument may be given as a two-digit number, where the first digit is the virtual screen and the second is the slot number on that screen. Screen 1 is the virtual screen currently displayed. Subsequent screen numbers are virtual screens to the right of the current virtual screen. So, for example slot '`31`' is interpreted as slot 1 on the virtual screen two to the right of the current screen. Single-digit slot numbers are on virtual screen 1.

The *rate* argument gives the time in seconds between data updates on the page. Pages are updated on a timed interval, which defaults to once every four seconds.

The *rate* argument has no effect on graphic pages (those with a `.disp` extension). The update rate of some page elements is specified when they are created. Others receive updates whenever a mnemonic is received.

Only a single instance of any given page may be displayed. If the named page already is running and the `page` directive is executed with no *slot* argument, the given page will be brought to the top of the window stacking order at its present location. If you give a *rate* ('`page`' *name*'`,`'*rate*), the page also will begin updating at the new rate. If a *slot* is given, the page will be moved to the new slot.

Use the `page clear` directive, the second form above, to remove pages from the display. If you give the keyword `ALL` rather than an individual page name, all pages are cleared except *base pages*. Base pages are those named by the mnemonics `gbl_basepg_1`, `gbl_basepg_2`, and `gbl_statuspg`, which typically are loaded from an '`itosrc`' file. Base pages may be closed with the mouse.

Use the `page freeze` directive to halt page updates, and use the `page thaw` directive to resume page updates. While a page is frozen, use the `page refresh` directive without a rate argument to do a one-time page update. Use the `page refresh` directive with a rate argument to change a running page's update rate. You also can do this with the first form of the directive, as previously mentioned.

PKTDUMP

The *PKTDUMP* directive allows users to display the contents of source packets as they arrive from the spacecraft or other data source.

`PKTDUMP [`*station*`] [`*option*`] [`*filter*`] [[> `*dest*`]...]`

Start a packet dump.

station	identifies the telemetry source.
option	ID = *id* COUNT = *count* BINARY
filter	`[ADD | DROP] [NOT]` [*vcspec* ...] *vcspec* ↦ `ALL |` *vcid* `[ADD | DROP] [NOT]` [*appid* [, *appid* ...] *vcid* ↦ `VC0 | VC1 | ... | VC7` *appid* ↦ `ALL |` integer (expression) application ID.
dest	is a dump destination: filename, printer (`ptr`[`:`*name*]), or display (`DSP` or `CRT`).

`PKTDUMP OFF` *option*

Stops a packet dump.

option	`| ALL` `| ID =` *id*

`PKTDUMP QUERY`

Reports on currently active packet dumps.

The `pktdump` directive starts a packet dump, which displays, prints, or captures in a file a set of raw telemetry packets given by *filter*. The packets are formatted into ASCII in a way that's reasonably easy to read, with all headers are broken out into individual fields, unless you specify the `binary` option.

Use the `binary` option may be used to capture packets in their original binary representation in a file. If given, only file output is honored; that is, output will not be routed to the display or printer.

Specify the *station* argument to select the data source providing the data you wish to dump, when more than one source is in use. Valid station names are in the Source Configuration File, and spacecraft data is the default source. When the data source is from an archive playback specify "playback" as the source.

Use the `id` option to name the packet dump. You give the same `id` to the `pktdump off` directive to stop that particular dump. If you don't specify an `id`, the system will provide one for you, which it reports in the event log.

The `count` option allows you to limit the number of packets you collect to *count*. After the given number is collected, the packet dump program exits, unless you are dumping to a display window. In that case, the program displays a '`Finished`' status and waits util you dismiss it.

The filter argument is as explained for the `acquire` directive, except that only one virtual channel may be specified. If no filter is given, all packets on virtual channel *gbl_tm_defvc* will be dumped.

Up to three optional dump destinations, *dest*, are permitted, with one each specifying output to a file, the display, or a printer. A file name may be an absolute pathname or simple file. In the latter case, the directory given by *gbl_pkdpdir* is prepended to the filename. A printer may be specified by giving a *dest* of the form `ptr[:`*printer*`]`, where the optional *printer* is a specific printer name. The default destination is the display, which may be specified explicitly by giving `dsp` or `crt` as *dest*.

PKTTIMEOUT

```
          | <interval> |  ALL
PKTTIMEOUT |            |
   | QUERY      | <pktids> [,<pktids>,...]
```

interval – is the time in seconds at which the packet static should be checked.

pktids – comma seperated list of packet numbers.

`PKTTIMEOUT QUERY ALL`
Reports the current timeout interval for all telemetry packets.

`PKTTIMEOUT QUERY pktids`
Reports the current timeout interval for the specified telemetry packets.

The `PKTTIMEOUT` directive modifies or retrieves the timeout interval for the specified telemetry packets `pktids`. A packet is flagged as static when it has not been received from telemetry stream within the timeout period.

```
PKTTIMEOUT ALL 10
⇒ All 109 tlm packets timeout interval set to 10 seconds.
PKTTIMEOUT 15 3,5,10
⇒ Packet id 3 timeout interval set to 15 seconds,
          Packet id 5 timeout interval set to 15 seconds,
          Packet id 10 timeout interval set to 15 seconds.
PKTTIEMOUT QUERY 3,6,10
⇒ Pktid  3 timeout interval set to 15 seconds,
          Pktid  6 timeout interval set to 10 seconds,
          Pktid 10 timeout interval set to 15 seconds.
```

PLAYBACK

The *PLAYBACK* directive replays previously archived telemetry.

Syntax

`PLAYBACK [`*options ...*`] [`*filter*`]`

`PLAYBACK QUERY`
: Query what playback is currently running. This option will override all others.

`PLAYBACK PAUSE`
: Pause the playback that is currently running.

`PLAYBACK RESUME`
: Resume a playback that is currently paused. If *rate* is currently zero it will be set to a maximum before the playback is resumed.

`PLAYBACK STOP`
: Stop the current playback.

option ↦ `SC`
option ↦ `GMT`
: Indicates wheter the start and stop times are spacecraft (`SC`) or wallclock (`GMT`) times. `GMT` is the default.

option ↦ `BEGIN =` *start_date*
option ↦ `END =` *stop_date*
: Start and stop dates.

option ↦ `RATE =` *rate*
: Playback rate in kilobits per second.

option ↦ `FILE =` *filename*
: Name of archive file to playback.

option ↦ `ID =` *id*

filter ↦ `[ADD|DROP] ALL`
filter ↦ `[ADD|DROP] [NOT] [`*vcspec ...*`]`
vcspec ↦ *vc*
vcspec ↦ *vc* `= [ADD|DROP ,] ALL`
vcspec ↦ *vc* `= [ADD|DROP ,] [NOT ,]` *appid* `[ ,` *appid ...*`]`
: Specifies which packets or transfer frames to play back.

Discussion

By default playback will look in the directory specified by *GBL_TM_ARCHDIR*. If you need playback to look in a different directory just explicitly enter the path to the filename. You may also set *GBL_TM_ARCHPATH* to a list of directories to search for an archive to playback. For a more intricate playback scenario start the archive replay GUI located in

the Progs section of the STOL window. When playback is started the global mnemonic *GBL_PLAYBACK_ACTIVE* is set to 1. When there is no playback running this global is set to 0.

An example of a simple playback:

```
playback file = "99215101112" rate = 23 vc0
```

An example of playback over a time range:

```
playback sc begin = "99-215-10:00:00" end = "99-216-10:00:00" rate = 23 vc0
```

An example of a playback started paused as in the example:

```
playback file = "99215101112" rate = 0 vc0 vc5
pktdump vc5
playback rate = 23
playback resume
```

This will allow the user to start an application such as PKTDUMP after the playback is started without missing any data. Enter PLAYBACK RESUME to start the data flow. If the rate is not set it will be reset to maximum before resuming the playback.

See Also

ARCHIVE.

PREVIEW

The PREVIEW directive controls STOL preview mode for handling input of directives from STOL interactive input.

Syntax

PREVIEW *ON*
: enables a two step directive mode.

PREVIEW *OFF*
: disables a two step directive mode.

PREVIEW *QUERY*
: displays the current state of two step directive mode

Discussion

With PREVIEW mode *OFF*, the operator is in a normal single step mode where a STOL_PREV is optional by pressing the Prev button in the STOL window. Pressing enter at any time will cause the directive to be executed. ASK boxes and Critical Alarm Popups also have a Preview button. In this mode the button is optional, the OK or SEND button are the default operation respectively.

With PREVIEW mode *ON*, the operator is required to perform a 2 step process of entering a directive in the STOL input box which will producre a STOL_PREV event message and then press the Exec button in the STOL window to cause the directive to be executed. One exception to this rule is a table of directives that don't require a STOL_PREV event message. This table is a global string mnemonic array called *GBL_STOLPREV_EXEMPT* which can contain up to 100 directive strings. If the directive entered in STOL is in this table then the preview is circumvented and immediately executed. Note: This mnemonic is only read once at startup by STOL so it must be initialized in the 'itosrc' file prior to starting ITOS. An example for the 'itosrc' fragment might be something like the following:

```
setenv ITOS_STOLPREV_EXEMPT_0_  "ac query"
setenv ITOS_STOLPREV_EXEMPT_1_  "acquire query"
setenv ITOS_STOLPREV_EXEMPT_2_  "ar query"
setenv ITOS_STOLPREV_EXEMPT_3_  "archive query"
setenv ITOS_STOLPREV_EXEMPT_4_  "cfgmon"
setenv ITOS_STOLPREV_EXEMPT_5_  "date"
setenv ITOS_STOLPREV_EXEMPT_6_  "dumpfile"
setenv ITOS_STOLPREV_EXEMPT_7_  "enable query"
setenv ITOS_STOLPREV_EXEMPT_8_  "global"
setenv ITOS_STOLPREV_EXEMPT_9_  "get"
setenv ITOS_STOLPREV_EXEMPT_10_ "gpib"
setenv ITOS_STOLPREV_EXEMPT_11_ "hotkey"
setenv ITOS_STOLPREV_EXEMPT_12_ "let"
setenv ITOS_STOLPREV_EXEMPT_13_ "limits"
setenv ITOS_STOLPREV_EXEMPT_14_ "log"
```

```
setenv ITOS_STOLPREV_EXEMPT_15_ "page"
setenv ITOS_STOLPREV_EXEMPT_16_ "pktdump"
setenv ITOS_STOLPREV_EXEMPT_17_ "pkttimeout"
setenv ITOS_STOLPREV_EXEMPT_18_ "playback"
setenv ITOS_STOLPREV_EXEMPT_19_ "plot"
setenv ITOS_STOLPREV_EXEMPT_20_ "preview query"
setenv ITOS_STOLPREV_EXEMPT_21_ "preview on"
setenv ITOS_STOLPREV_EXEMPT_22_ "purge"
setenv ITOS_STOLPREV_EXEMPT_23_ "quit"
setenv ITOS_STOLPREV_EXEMPT_24_ "rem"
setenv ITOS_STOLPREV_EXEMPT_25_ "seqprt"
setenv ITOS_STOLPREV_EXEMPT_26_ "set"
setenv ITOS_STOLPREV_EXEMPT_27_ "sho"
setenv ITOS_STOLPREV_EXEMPT_28_ "shoval"
setenv ITOS_STOLPREV_EXEMPT_29_ "snap"
setenv ITOS_STOLPREV_EXEMPT_30_ "speed"
setenv ITOS_STOLPREV_EXEMPT_31_ "start query"
setenv ITOS_STOLPREV_EXEMPT_32_ "stripchart"
setenv ITOS_STOLPREV_EXEMPT_33_ "system"
setenv ITOS_STOLPREV_EXEMPT_34_ "tfdump"
setenv ITOS_STOLPREV_EXEMPT_35_ "timeout"
setenv ITOS_STOLPREV_EXEMPT_36_ "zero"
```

Also in the 2 step mode the ASK boxes have the PREVIEW button as manditory and become the default button. The OK or SEND buttons become deactivated until the PREVIEW button is pressed. Once the the preview message goes into the event log the OK or SEND buttons become operational and will function as normal.

PLOT

The PLOT directive controls plotting of live data and data stored in files.

Syntax

PLOT *plotname* [, *rate*] [< *data*] [>[>] *output* ...]

Starts a plot of live data or data from a file.

plotname	is the name of the plot to start.
rate	is the update rate.
data	is the name of the data input file.
output	is the name of a graphical or data output file.

PLOT CLEAR *plotname* | ALL

Clears a plot from the display; kills a plot.

Discussion

The plot directive controls the ITOS plotting capabilities. In its first form above, it starts a plot operation according to the instructions in the plot definition file *plotname*.plot. The system looks for plot definition files in directories listed in the page path given in (*GBL_PAGEPATH*).

If a *data* file name is given, data is read from that file and plotted; otherwise, data is read from the telemetry stream in real time. For real-time plots, the plot definition must reference only valid telemety mnemonic names in its x and y value expressions. When plotting file data, the symbols used in the value expressions should correspond to column headings in the data file. See See section "value stmnt" in *ITOS Plotting Users' Guide*, for more information.

Up to two optional *ouptut* file names are permitted, one specifying a graphical output file and the other specifying a data output file. The suffix of the filename determines which type of file is being given: Data output filenames must end in '.dat'. Graphical output filenames may end it either '.xwd' or '.gif' In the former case, an X window dump file is produced; in the latter case a Graphics Interchange Format (GIF) image file is produced. Graphical output also may be directed to the printer by giving an output file name of the form ptr[:*printer*], where the optional *printer* is a specific printer name.

Currently, the data file output **is not supported**. When it is supported, it obviously will be useful only for real-time data.

If a graphical output file is specified for a plot of file data, the plot program will exit automatically after producing the output file. If you're plotting live data, the program will use the given output file name as the default name for snaps you command with the mouse.

At some future time, we plan to add the ability to periodically auto-snap real-time plots, but this is not yet available.

The *rate* option allows you to do real-time plots on a timed interval. Normally, real-time plots are updated whenever a new value comes in to the system. If the *rate* option is given, however, the plot will update all charts using currently available values every *rate* seconds.

Use the `plot clear` form of the directive to clear a plot from the display and kill the plot program. Specify `all` instead of a plot name to clear all plots.

PROC

```
PROC name [ (parmlist) ]
   .
   .
   .
ENDPROC
```

name – the name of the proc.

parmlist – comma-seperated list of names. Each name in `parmlist` becomes a local variable whose initial value is the value of the corresponding argument in the `START` directive (see [START], page 90).

The `PROC` directive is only permitted as the first directive in a proc file (see section "Proc Files" in *ITOS STOL*); any other usage results in a syntax error. The `PROC` directive identifies the arguments to the procedure.

See [RETURN], page 76, for how to terminate a proc file. See [START], page 90, for how to invoke a proc file.

PURGE

```
PURGE
```

This directive may optionally have a preceding slash, as in `/PURGE`.

This directive is only allowed when commanding is enabled (see [ENABLE], page 24).

The `PURGE` directive removes all commands from the Sent-Queue which are pending acknowledgement from the spacecraft

QUIT

```
QUIT
```

The `QUIT` directive terminates `olstol`.

RAW

`RAW hex`

hex – hexadecimal numbers. Each number consists of two hexadecimal digits and represents one octet of the command. Pairs of hexadecimal digits may be seperated by spaces or commas. The following are some of the ways to specify the eight-octet source packet `0x0001020304050607`:

```
RAW 00 01 02 03 04 05 06 07
 - or -
RAW 0001020304050607
 - or -
RAW 00010203, 04050607
 - or -
RAW "000102", (1+2), 040506, (263)
```

This directive may optionally have a preceding slash, as in `/RAW ...`.

This directive is only allowed when commanding is enabled (see [ENABLE], page 24).

The `RAW` directive sends a spacecraft command specified in raw hexadecimal at the source CCSDS Telecommand packet level. See section "Spacecraft Commands" in *The ITOS Command Subsystem*.

RAWTF

`RAWTF hex`

hex – hexadecimal numbers.

This directive may optionally have a preceding slash, as in `/RAWTF ...`.

This directive is only allowed when commanding is enabled (see [ENABLE], page 24).

The `RAWTF` directive sends a spacecraft command specified in raw hexadecimal at the CCSDS transfer frame level. See section "Spacecraft Commands" in *The ITOS Command Subsystem*.

The `RAWTF` directive assumes a CCSDS Telecommand Packet within a CCSDS Telecommand Transfer Frame. If *gbl_chksumrout* is set, a checksum is appended to the packet within the transfer frame. To disable appending the checksum, set `GBL_CHKSUMROUT=""`.

The Telecommand Transfer Frame header's frame length field determines to size of the `RAWTF`. Excess octets in the `RAWTF` directive are ignored; if there are too few octets in the `RAWTF` directive, the missing values in the resulting transfer frame are unpredictable.

RRAW

`RRAW hex`

hex – hexadecimal numbers. Each number consists of two hexadecimal digits and represents one octet of the command. Pairs of hexadecimal digits may be seperated by spaces or commas. The following are some of the ways to specify the eight-octet source packet `0x0001020304050607`:

```
RRAW 00 01 02 03 04 05 06 07
 - or -
RRAW 0001020304050607
 - or -
RRAW 00010203, 04050607
 - or -
RRAW "000102", (1+2), 040506, (263)
```

This directive may optionally have a preceding slash, as in `/RRAW ...`.

This directive is only allowed when commanding is enabled (see [ENABLE], page 24).

The `RRAW` (a.k.a. Really Raw) directive sends a spacecraft command specified in raw hexadecimal at the source packet level. This directive is the same as the `RAW` directive but without the CCSDS Telecommand headers. See section "Spacecraft Commands" in *The ITOS Command Subsystem*.

READ

```
READ ( unit [ options ...] )
 - or -
READ ( unit [ options ...] ) variable
 - or -
READ ( unit [ options ...] ) variable , variable ...
```

unit – int constant, variable, or expression identifying the unit to be read.

variable – identifies a local variable, global variable, or mnemonic.

Options are space seperated and may be specified in any order and include:

`DELIM=`*delimiters*
: Indicates the delimiters that seperate words. Defaults to `",\t "` (comma, tab, or blank).

`GREEDY`
: Saves the rest of the line in the final variable.

  Normally, delimiter-separated words get saved; if there aren't enough variables trailing words get lost. Specifying the `GREEDY` option changes this behavior so that the final argument gets the rest of the line.

  If there are too many variables, i.e. if there are *m* variables in the `READ` directive but only *n* words in the input line, the final *m-n* variables are unchanged.

`STATUS=`*variable*
: Specifies a local or global variable or integer mnemonic to receive the status from the read operation. States are 1 or greater == SUCCESS, 0 == END-OF-FILE, -1 == ERROR. If SUCCESS then the value of `STATUS` will contain the number of variables minus 1 that had values applied. i.e a `STATUS` value of 2 indicates 1 variable was set. A value of 1 indicates no values were set or in other words the line was blank or no variables were given on the directive line.

  It is up to the user to test the return *variable* before attempting another READ. If STATUS is not specified, the proc will stop with an error condition if the status is anything other than SUCCESS.

`TIMEOUT=`*seconds*
: Indicates how long the READ may wait before timing out. Defaults to `1.0` second, `0.0` waits forever. An interactive `GO` directive causes the `READ` to immediately time out.

  Use the following trick to determine if the `READ` timed out:

```
local x, stat
x=0 ; force x to be int rather than string
read(1 TIMEOUT=4 STATUS=stat) x
if (isint(x)) then
   rem ; The read timed out (or the line was empty)
else
```

```
            rem ; x was read
            if (stat .EQ. 0) then
               rem ; End of File was reached
            endif
         endif
```

For example, if '`./test.data`' contains

```
one two three
four five six
seven eight
nine ten eleven
```

then:

```
GLOBAL i,j,k
OPEN (1) "./test.data"
READ (1)
⇒ "one two three" was read.
READ (1) i,j
⇒ "four five six" was read.
⇒ Assigning (string) 'four' to global variable i.
⇒ Assigning (string) 'five' to global variable j.
k = 99
⇒ Assigning (int) 99 to global variable k.
READ (1) i,j,k
⇒ "seven eight" was read.
⇒ Assigning (string) '7' to global variable i.
⇒ Assigning (string) 'eight' to global variable j.
⇒ Warning -- k is unchanged.
READ (1 GREEDY) i,j
⇒ "nine ten eleven" was read.
⇒ Assigning (string) 'nine' to global variable i.
⇒ Assigning (string) 'ten eleven' to global variable j.
```

See [OPEN], page 49 for how to open a unit. See [WRITE], page 107 for how to write to an open unit.

REM

```
REM ; comment
```

The `REM` directive doesn't do anything. It's only purpose is to provide a way for proc files to echo comments to the event log:

```
; This comment won't get echoed to the event log ...
REM ; ... but this one will!
```

RESET

`RESET`

This directive may optionally have a preceding slash, as in `/RESET`.

This directive is only allowed when commanding is enabled (see [ENABLE], page 24).

The `RESET` directive sends the control command to the spacecraft FARM which sets the next expected frame sequence number to zero.

Note that control commands are transmitted immediately, even when in two step command mode. See [MODE], page 48.

See [UNLOCK], page 102.

RESYNC

`RESYNC`

This directive may optionally have a preceding slash, as in `/RESYNC`.

This directive is only allowed when commanding is enabled (see [ENABLE], page 24).

The `RESYNC` directive sets the ground's next expected fram sequence number to the spacecraft's next expected frame sequence number.

RETRLIM

`RETRLIM number`

This directive may optionally have a preceding slash, as in `/RETRLIM 3`.

This directive is only allowed when commanding is enabled (see [ENABLE], page 24).

The `RETRLIM` directive sets the maximum allowable number of automatic retransmissions of a command to the spacecraft before operator intervention is required. `RETRLIM 0` turns off automatic retransmission.

RETRY

`RETRY`

This directive may optionally have a preceding slash, as in `/RETRY`.

This directive is only allowed when commanding is enabled (see [ENABLE], page 24).

The `RETRY` directive retransmits all commands in the Sent-Queue pending acknowledgement by the spacecraft.

RETURN

```
RETURN
```

```
RETURN HALTED
```

The `RETURN` directive returns from and terminates the current proc.

In addition, `RETURN HALTED` for a sub proc causes the calling proc to halt.

Use the `KILLPROC` directive to terminate all procs (see [KILLPROC], page 40).

SCEVENT

```
SCEVENT severity, evtnum [ options ]
 - or -
SCEVENT severity, evtnum [ evtstr ]
```

severity – int constant, variable, or expression identifying the event severity of 0 to 255.

evtnum – int constant, variable, or expression identifying the event number of 0 to 65535.

evtstr – string constant, variable, or expression identifying the event description. Same as the option DATASTR=*string* below.

Options may be specified in any order and include:

`DATA1=`*integer*
: Specifies an optional 32 bit integer constant, variable, or expression value. Default is 0.

`DATA2=`*integer*
: Specifies an optional 32 bit integer constant, variable, or expression value. Default is 0.

`DATA3=`*integer*
: Specifies an optional 32 bit integer constant, variable, or expression value. Default is 0.

`DATA4=`*integer*
: Specifies an optional 32 bit integer constant, variable, or expression value. Default is 0.

`DATASTR=`*string*
: Specifies an optional string constant, variable description of the event. Default is NULL.

This directive is effectively a NO-OP and is only syntax checked for correctness. It is only for the purpose of consistancy with STOL procs running on the Triana spacecraft and no other purpose.

SEND

`SEND`

This directive may optionally have a preceding slash, as in `/SEND`.

This directive is only allowed when commanding is enabled (see [ENABLE], page 24).

SEQPRT

The *SEQPRT* directive controls sequential prints, which output telemetry data in tables to files, the display, a printer, or the event log.

Syntax

`SEQPRT` *seqprt*[, *interval*] [[`>`]`>` *file*] [`>` *dest* ...]

Start a sequential print.

seqprt ↦ is a sequential print name, which is the definition file name without the '`.sprt`' extension.

interval ↦ is the interval at which data should be printed. If not given, data is printed as it arrives in telemetry, according to rules given in the definition file.

file ↦ is the name of the output file. If preceeded by a single greater-than symbol (`>`), create or overwrite the file. If preceeded by two greater-than symbols (`>>`) create or append to the file.

dest ↦

printer ↦ `PTR | PRT | PRN`

display ↦ `DSP | CRT`

event log ↦ `EVT`

`SEQPRT` `CLEAR` *seqprt* `|` `ALL`

Stop a sequential print, or stop all sequential prints.

`SEQPRT` *action seqprt*

Control the updating of a sequential print.

action ↦

`FREEZE`	Stop the print from updating.
`REFRESH`	Cause the print to update immediately, even if frozen.
`THAW`	Allow the print to resume updating.

seqprt is the sequential print name.

Discussion

The actual sequential print description file is in one of the directories in the *gbl_pagepath* and is named with the suffix `.sprt`.

Normally sequential prints update whenever a value in the sequential print changes; specifying *interval* changes this behavior to updating every *interval* seconds.

SET

```
SET device pid [= par]
```

device – A device (BITSYNC, PSK, RECORDER, RECEIVER, etc.).

pid – Parameter identification. The name of the particular function being programmed.

=*par* – Optional parameter value for *pid*.

Examples:

```
SET BITSYNC BR=8.192e03

SET BITSYNC SOURCE1

SET RECORDER REPROPBN=9999
```

The SET directive is used to send commands to various devices capable of being controlled remotely over a GPIB or RS-232 interface. The definition of devices and their commands are in the file Device.conf. This directive relies on the settings of global mnemonics to determine which model of a particular device is being used and the communications setup parameters required for the device. For example, the Device.conf configuration file may define the commands for two different external devices, a bit synchronizer and an oscilliscope (eg.Loral DBS 430, and TEKTRONICS 3430). The global mnemonics for the devices, found in *ITOS_GPIB*, specify which device is being used such as:

```
GBL_DEV_TYPE[0]  = BITSYNC
GBL_DEV_MODEL[0] = LORALDBS430
GBL_DEV_IF[0]    = RS232
GBL_DEV_PORT[0]  = /dev/ttya
GBL_DEV_BAUD[0]  = 9600
GBL_DEV_CLEN[0]  = 8
GBL_DEV_SBIT[0]  = 1

GBL_DEV_TYPE[1]  = TEKSCOPE
GBL_DEV_MODEL[1] = TEK2430
GBL_DEV_IF[1]    = GPIB
GBL_DEV_PORT[1]  = GPIB-ITOS
GBL_DEV_ADDR[1]  = 4
```

A default Device.conf file comes with ITOS. The user can modify this file or create their own Device.conf and set the global *gbl_devcfgdir* to the directory where it is located. This is an example of the entries that need to be added to the Device.conf file in order to use the GET and SET directives.

```
$PIDS                  # DO NOT ERASE THIS LINE
#
# PID         FIXED/VARIABLE     Comment
#-------------------------------------------------
```

```
SOURCE1,      FIXED              # Input source 1
BR,           VARIABLE           # Bit rate.
REPROPBN,     VARIABLE           # Reproduce from PBN
$END                  # DO NOT ERASE THIS LINE - ADD CMDS BEFORE THIS LINE

$MODEL=DSI7700
#
# Bit Synchronizer, Model DSI 7700
#
# PID:PAR Pair ASCII Cmd String,   Comment
#----------------------------------------
SOURCE1,      "SRC1."               # Input source 1
BR,           "B%9v999\.E9%."       # Bit rate.
REPROPBN,     "%999999%"            # Reproduce from PBN
$END
```

For VARIABLE parameters such as bit rate(BR) shown above a COBOL-style PIC format enclosed within %'s is used and is described below.

```
The PIC may contain the following characters:

      '+', '-' :  The formatted number will begin with a
                  '+' or '-'.

      '9'      :  Used to specify field width.  Each '9' is
                  used to represent a digit.

      '.'      :  Used for floating point numbers to specify
                  decimal point placement.

      'E','e'  :  Specifies the number to be formatted using
                  exponentiation using the letter 'E'.

      'Z','z'  :  Whenever a 'Z' appears before the digit
                  specifier, leading zeros will be suppressed.
                  By default, the field width is padded with
                  zeros.

      'V','v'  :  Used to show placement of assumed decimal
                  point.

      '\'      :  Precedes literal characters in the formatted
                  number string.  At this time literals may
                  only be placed immediately preceding the
                  exponent character 'E'.

PIC Examples:

    NUMBER        PIC                FORMATTED NUMBER
    ------------------------------------------------------
```

```
12345          +999999.9          +012345.0
12345          Z999999.Z9         12345.
12345          9999               2345
12345.67       Z9.999999E+999     1.234567E+004
12345          9.999EZ99          1.234E4
123.45         9.9999E99          1.2345E02
123.45         9.9E9              1.2E2
123456         9v999\:E9          1234:E5
```

Also see [GET], page 30, [ENABLE], page 24 and [DISABLE], page 18.

SETCOEF

```
SETCOEF mnemonic, c0 [, c1 [, c2 [, c3 [, c4 [, c5 [, c6 [, c7]]]]]]]

SETCOEF QUERY mnemonic [, ...]
```

mnemonic – telemetry mnemonic name.

c0 ... *c7* – floating point constants or expressions.

The `SETCOEF` directive reports or changes analog conversion coefficients.

`SETCOEF CHANGE`

Unspecified coefficients are assumed `0.0`. The analog conversion function is $c_7X^7 + c_6X^6 + c_5X^5 + c_4X^4 + c_3X^3 + c_2X^2 + c_1X + c_0$.

`SETCOEF QUERY`

Reports the current analog conversion coefficients for the specified mnemonics. The report looks like:

```
PAHICURR: 0.004884X + -10
 - or -
PBATTEMP: 7.11667e-08X^3 + -0.00013295X^2 + 0.1352X + -65.113
 - or -
GBL_TLMRATE: (polynomial undefined)
 - or -
GBL_GMTOFF isn't analog; it's conversion type is DATE
```

SETGRND

`SETGRND number`

This directive may optionally have a preceding slash, as in `/SETGRND 49`.

This directive is only allowed when commanding is enabled (see [ENABLE], page 24).

The `SETGRND` directive sets the ground's next expected frame sequence number to the operator specified value.

SETVR

`SETVR number`

This directive may optionally have a preceding slash, as in `/SETVR 33`.

This directive is only allowed when commanding is enabled (see [ENABLE], page 24).

The `SETVR` directive sets the spacecraft's and ground's next expected frame sequence number to the operator specified value. This directive will send a control command to the spacecraft.

SHOVAL

```
SHOVAL value [, value ...]
```

SHOVAL may be abbreviated SHO.

The SHOVAL directive displays the values of one or more variables, telemetry mnemonics, or expressions.

If value is a variable or telemetry mnemonic name (i.e., not a complex expression and not enclosed in parenthesis) the output describes the variable or telemetry mnemonic in addition to displaying its value.

In the following examples, myvar is a global variable whose value is 7.89 and APRECOILCURR is a telemetry mnemonic:

```
SHOVAL myvar
⇒ global myvar: (double) 7.89
SHOVAL (myvar)
⇒ 7.89
SHOVAL 1, 1+2, 1+2+3, 1+2+3+4
⇒ 1 3 6 10
SHOVAL aprecoilcurr
⇒ mnemonic APRECOILCURR: (NOVALUE) UNSIGNED 0
SHOVAL format("%08x",-1)
⇒ ffffffff
```

SIM

```
                                | packet |
SIM CHG mnemonic = value [, |         |]
                                | ALL    |
```

mnemonic – the name of a telemetry mnemonic.

value – a constant or expression.

packet –

The `SIM` directive controls the internal simulator.

SNAP

```
SNAP page [, rate] [>|>> destination]

           | page
SNAP CLEAR |
           | ALL
```

page – the page to be snapped.

rate –

destination –

> *destination* –

>> *destination* –

The SNAP directive controls page snapshots.

SNAP PAGE The SNAP PAGE directive takes a snapshot of a page, or starts a snap process that repeatedly takes snapshots of a page.

If rate is not specified only one snapshot is taken. If rate is specified, a snap process will be started which takes a snapshot every rate seconds.

If destination is not specified the snapshot will be printed to the default system printer.

SNAP CLEAR

SPEED

```
SPEED rate
SPEED rate PROC name
SPEED rate ALL
SPEED rate DEFAULT
SPEED QUERY
SPEED QUERY PROC name
SPEED QUERY ALL
SPEED QUERY DEFAULT
```

> `rate` – floating point number, or keyword `DEFAULT`. `DEFAULT` makes the proc run at *gbl_defprocspeed*.

The `SPEED` directive controls (or reports) how fast proc files execute.

`SPEED rate` changes the speed of the current proc to one directive every `rate` seconds. For example, `SPEED .1` changes the current proc's speed to one directive every `.1` seconds.

Similary, `SPEED rate PROC name` changes the speed of the named proc and `SPEED rate ALL` changes the speed of all open procs.

`SPEED rate DEFAULT` changes the default proc speed (and thus the speed of all procs running at default proc speed) (`SPEED DEFAULT DEFAULT` is nonsensical and is not allowed).

Finally, `SPEED QUERY` reports the speed of the current proc; `SPEED QUERY PROC name` reports the speed of the named proc; `SPEED QUERY ALL` reports the speed of all open procs; and `SPEED QUERY DEFAULT` reports the default proc speed.

Procs initially start with the default proc speed *gbl_defprocspeed*.

START

```
START proc [( arglist )] [IN path] [AT line] [HALTED] [&]

START QUERY
```

The START directive starts a proc file or queries for the status of foreground and background procs.

proc specifies the name of the proc without the ".proc" or it can specify the fully qualified path name of the proc file such as "/home/mission/procs/startup.proc" which can be used in place of using the "IN path" argument.

(arglist), IN path, AT line, and HALTED are all optional but if specified must be specified in the order shown.

(arglist) is the comma separated argument list to the proc. The number of arguments in arglist much match the number of arguments in the proc file's PROC directive (see [PROC], page 63) or you will get a warning message and the missing arguments will be set to NULL. NULL paramters can be tested by using the ISNULL function call.

IN path specifies the proc file to use. If IN path is not specified, the directories in *gbl_procpath* are searched.

AT line specifies which line in the proc file to begin at. line can be either a line number or a label.

HALTED specifies that the proc should be initially halted.

"&" specifies that the proc starts as a background task in a sub-stol. WARNING: Telemetry, Command and GPIB directives should not be mixed between foreground and background procs simultaneously because they could interfere with each other. Especially when doing Command directives, the proper technique is for a background proc to send these directives via a [WRITE], page 107 directive to the *gbl_stolfifo* to the main STOL for execution. The sub-stol and background proc windows are instantiated as icons rather than an open window so as not to clutter the desktop. STOL directives and most information messages are not echoed to the event log/display but fault and error messages are. A background proc can not start another background proc. The spawned sub-stol process number is placed in global mnemonic *gbl_system_pid* which can be used later to kill that process via [KILLPROC], page 40 directive. This is only a basic function of parallel proc processing and lacks sophistication. For example, the following proc fragment to start a background proc

```
GLOBAL pid

start test &
==>STOL_MSG: Background PROC "test" (ID# 1234) started.
==>STOL_MSG: Opened test in /home/itos/procs/test.proc
pid = gbl_system_pid

start query
==>STOL_MSG: No foreground procs are open
==>STOL_MSG: Background PROC "test" (ID# 1234) Running
```

See [KILLPROC], page 40 for how to terminate a proc.

STOP

`STOP`

This directive may optionally have a preceding slash, as in `/STOP`.

This directive is only allowed when commanding is enabled (see [ENABLE], page 24).

The `STOP` directive aborts the image load currently in progress. If in TWOSTEP mode, the command buffer will be cleared.

See [LOAD], page 44. See section "Image Loads" in *The ITOS Command Subsystem*.

STRIPCHART

The STRIPCHART directive allows the user to chart three kinds of data:

1. Live data arriving less than once a second.

   STRIPCHART [*expression* ...] [*option* ...]

2. Live data arriving more often than once a second.

   STRIPCHART *expression* [...] FAST [*option* ...]

3. Data stored in a sequential print file.

   STRIPCHART *label* ... FILE=*path* [*option* ...]

expression is any stol expression, enclosed in quotes. Up to four expressions may be listed. In the first form above, expressions are optional. If none is specified, a dialog is presented where expressions can be entered. In the second form, all expressions must be specified in the STRIPCHART directive.

option is any combination of the options discussed below.

label is any label in the first line of the sequential print file. One to four labels may be plotted.

path is the absolute path to the sequential print file to be plotted.

Three commands modify onscreen stripcharts:

1. Stop data to a stripchart

   STRIPCHART FREEZE *name*

2. Restart data to a stripchart

   STRIPCHART THAW *name*

3. Close a stripchart

   STRIPCHART CLEAR *name*

Names are given to stripcharts with the NAME option described below. *name* can be "ALL" in these three commands.

Discussion

The `stripchart` directive allows charting without prior creation of a plot definition file (a `.plot` file) or a graphics page file (a `.disp` file). The PLOT directive uses `.plot` files. makepage can be used to create graphics page files used by the PAGE directive.

The following options are recognized:

BACKGROUND = *color*
: Sets the chart's background color. Default is black.

FOREGROUND = *color*
: Sets the chart's foreground color. Default is white.

`FAST` Starts a chart optimized for high-data-rate mnemonics. This option should be used with any mnemonic received once a second or more.

This chart is less flexible than non-optimized charts. Maximum and minimum values for the Y axis should be specified with YMAX and YMIN options when the chart is created. Non-optimized charts adjust the Y axis as data arrives. (See "Fast Options" below.) The MAXPOINTS option is not recognized by fast charts. The x expression must be a time mnemonic. In other kinds of plot, any STOL expression can be used for the x expression.

`HEIGHT =` *pixels*
Sets the chart's height. Default is 320 pixels.

`HOST =` *name*
specifies the host from which live data should be obtained. The default is the local host.

HOST may not be combined with FILE. See below.

`LEFT =` *pixels*
sets the distance from the left side of the screen to the left side of the chart. Default is 0.

`LEGEND =` *on/off*
specifies whether a legend is included with the chart. By default a legend is supplied if the chart contains more than one graph.

`MAXPOINTS =` *points*
sets the maximum number of points which will appear on the chart. When points are added over the maximum number, the oldest points are removed. The default value is 200 points. Note that this number can also be changed while a chart is being displayed via the "options" button.

MAXPOINTS is ignored with FILE and FAST. With FILE, all points in the file appear on the chart. With FAST, the width of the chart together with SLOTWIDTH (see below) determine how many points appear on the chart.

`NAME =` *name*
Gives the chart a name that can be used in FREEZE, THAW and CLEAR directives. All stripcharts onscreen have different names. If you try to create a second chart with a name already in use, the first chart is popped to the top and no new chart is created. The default if no name is specified is the text of the first expression.

`PORT =` *number*
specifies the server socket from which live data will be obtained. The default is 7777, the data point server's default.

PORT may not be combined with FILE. See below.

`PRINT =` *on/off*
specifies whether a print button is supplied with the chart. Default is *on*.

`TOP` = *pixels*
: Sets the distance from the top of the screen to the top of the chart. Default is 71 pixels, just enough to leave the STOL window exposed.

`WIDTH` = *pixels*
: Sets the width of the chart. Default is 600 pixels.

`X` = "*expression*"
: Specifies the expression used to generate x coordinates for plotted points. Any STOL expression, enclosed in quotes, may be used. The default is P@GBL_GMTOFF, the converted value of *GBL_GMTOFF*.

  When used with the FILE option, the expression must be a label from the sequential print file.

  This option cannot be used when FAST is specified. The TIME option, described below, is used instead.

`YMAX` = *max*
: Specifies the value at the top of the Y axis. By default, the Y axis autoscales to fit the data. When FAST is specified, the default is 100.

`YMIN` = *min*
: Specifies the value at the bottom of the Y axis. By default, the Y axis autoscales to fit the data. When FAST is specified, the default is 0.

The following options are recognized only with the FAST option:

`SLOTDELTA` = *sec*
: The chart area is divided into vertical slots and all data points is placed in the middle of a slot. *sec* specifies the time difference between adjacent slots. The default is 0.1 seconds.

`SLOTWIDTH` = *pixels*
: Specifies the width of slots in pixels. The default is 3 pixels.

`TIME` = "*mnemonic*"
: Specifies the time mnemonic to be used as the X value. The default is *GBL_GMTOFF*. NOTE that the mnemonic name must be enclosed in quotes.

`YDELTA` = *delta*
: Labels along the Y axis will be *delta* apart. If *delta* is 0.5 and *ymin* (see above) is 13.0, then the bottom three labels on the Y axis are 13.0, 13.5 and 14.0.

  This option is needed only when the default labels are not satisfactory.

`YHASHDELTA` = *hashdelta*
: Specifies how far apart the Y axis hash marks are placed.

  This option is needed only when the default hash marks are not satisfactory.

Use the `stripchart file=` form of the directive to view data from a file created by the SEQPRT directive. Each expression to be plotted must be a label from the sequential print file. At least one label from the file must be given.

The X option is used to specify the file label that will be used for x coordinates. If X is not used, the first column in the file is used.

The following option is recognized only when FILE is present:

XNOTIME means that the X variable is not a time variable. By default, X values are converted to time values.

The following options are NOT allowed when FILE is present:

MAXPOINTS
All points in the file are charted. The chart can be zoomed to an area of interest by dragging the left moust button.

HOST does not apply when data is taken from a file.

PORT does not apply when data is taken from a file.

TIME Use the X option rather than TIME to specify the X axis label.

Examples

```
STRIPCHART
```

Opens an empty chart window. You enter expressions to be plotted via the chart's option menu.

```
STRIPCHART psbatcurr psbatvolt "psbatcurr * psbatvolt" MAXPOINTS=32
```

Charts the two mnemonics and their product. Only the most recent 32 points are shown. Notice the quotes around the expression. The simple mnemonic expressions psbatcurr and psbatvolt can be quoted as well, but that's not required. Since no time mnemonic is specified, gbl_gmtoff will be used.

```
STRIPCHART "p@psbatcurr" "p@psbatvolt" X="h00btime"
```

Charts the converted values of the two mnemonics against the time mnemonic h00btime. Notice the required quotes around both expressions and the time mnemonic's name.

```
STRIPCHART "sin(p@tovolts)" FAST YMIN=-1 YMAX=1 SLOTDELTA=1.0
```

Charts a single expression. Since FAST is used, the y axis minimum and maximum values are specified. Data is expected every second.

```
STRIPCHART tvp5 tvp6 tvp7 tvp8 FILE="/usr/tcw.trace/archive/thermal_data.sav"
```

Charts values from file `thermal_data.sav`. Data in the columns headed by `tvp5`, ..., `tvp8` are charted. Since the `X` option is not used, the first column in the file (other than `tvp5`, ..., `tvp8`) is used. That first column is typically time.

SYSTEM

```
SYSTEM unix-command
```

`unix-command` – A string constant or expression representing a unix command.

The unix command will be processed via `sh` rather than `csh`.

Runs the specified unix command/program in the background; results go in the event log as STOL_EVENT messages. The mnemonic variable *GBL_SYSTEM_PID* receives the process id of the unix process started which can be used later to kill that process by a `SYSTEM concat("kill ", gbl_system_pid)`. Kill is a special SYSTEM directive. It will only allow you to kill a process started by a previous SYSTEM directive. If an error occurs trying to issue the SYSTEM directive, the return value in *GBL_SYSTEM_PID* will be "-1". Kill can be passed an optional signal number before the process number in the form of "n" or "-n" where `n` is a valid signal number. The default signal if none is specified is "9" which is *SIGKILL* which forces a absolute immediate exit of a process. A usefull signal is "15" which is a *SIGTERM* which causes a gracefull shutdown of a process but the process might be blocked or ignore it. See examples below. See Unix Man pages on `KILL` and `SIGNAL` (section 7 on Linux and 5 on Solaris) for the list of valid signals. Note: only the signal number can be used not the symbolic name.

Some examples:

```
LOCAL device
device = "/dev/ttyb"
SYSTEM concat("flush ",device)
```

flushes pending input and output serial port /dev/ttyb;

```
SYSTEM date
```

reports the current date (i.e. `Fri Apr 29 18:25:58 EDT 1994`); and

```
SYSTEM "xterm -T \""TEST\"" -fn 12x24"
```

opens a big font xterm window named TEST. (The quotes are required since the command contains special chars. If quotes have to be embedded in the string as in the example above, double quotes have to be used to make `STOL` understand and a `\` in front of the double quotes to make `UNIX` not interpret the quotes.

```
local pid
SYSTEM "xterm"
pid = gbl_system_pid
SYSTEM concat("kill ", pid)
```

Start an xterm window then later Kill the process started by the last system call.

```
SYSTEM concat("kill -0", pid)
```

The alternate kill example above uses an optional signal number of "0" which will not kill the process. If the process still exists then *GBL_SYSTEM_PID* will contain the process number in the kill. It it does not then *GBL_SYSTEM_PID* will equal "-1". This is a usefull test to see if a process is still running.

TFDUMP

The *TFDUMP* directive allows users to display the contents of telemetry frames as they arrive from the spacecraft or other data source.

`TFDUMP [`*station*`] [`*option*`] [`*filter*`] [[> `*dest*`]...]`

Start a frame dump.

station identifies the telemetry source.

option
ID = *id*
INT = *interval*
COUNT = *count*
BINARY

filter
`[ADD | DROP] [NOT]` [*vcid* ...]
vcid ↦ `VC0 | VC1 | ... | VC7`

dest is a dump destination: filename, printer (`ptr`[:*name*]), or display (`DSP` or `CRT`).

`TFDUMP OFF` *option*

Stops a packet dump.

option
`| ALL`
`| ID =` *id*

`TFDUMP QUERY`

Reports on currently active packet dumps.

The `tfdump` directive controls transfer frame dumps. A transfer frame dump is similar to a packet dump (see [PKTDUMP], page 54), the difference being that the transfer frame dump displays entire transfer frames.

Specify the *station* argument to select the data source providing the data you wish to dump, when more than one source is in use. Valid station names are in the Source Configuration File, and spacecraft data is the default source. When the data source is from an archive playback specify "playback" as the source.

All options are arguments and options are as documented for `pktdump` directive, with the exception of the *filter* argument. For transfer frame dump, the *filter* may contain multiple virtual channels, but must not contain any packet ID lists.

TIMEON

`TIMEON datafile [,[threshold] [,time]] [>|>> dest]`

`datafile` – input/output file name.

`threshold` – threshold file name.

`time` – maximum gap between consecutive dates.

`dest` – output file name.

`TIMEON` computes the time various spacecraft devices have been powered up daily and in total.

`TIMEON` reads a table of values from `file`. The first column in the table must contain date values. It optionally reads a 1-column table of thresholds from file `threshold`. There should be one threshold for each column (in order) in the table, not including the date column. Note that either or both files may be compressed (with `gzip`).

The results are written back to the data file, replacing the original contents, unless another output file, `dest`, is specified. Before the original data file is overwritten, however, it's contents will be compressed and copied to a backup file. The backup file name is formed from the original file name, appending a date stamp corresponding to the first data row in the file and a serial number beginning with zero. Backup files will have the '`.gz`' suffix because they are compressed with `gzip`.

For each row after the first, `TIMEON` computes the difference between the current row's date and the previous row's date. For each column in the row except the date column, if the value is greater or equal to the threshold, the device is powered. If the preceding reading showed the device powered, it adds the difference in the dates to the daily and grand totals for the column.

`TIMEON` keeps track of the day in which each date falls. When it encounters a new day, it replaces the previous day's table entries with a single entry giving the date and total times on for each device for that day. If, while processing, it encounters a time value, it adds the time to the daily and grand totals for the column.

For example:

If we begin with this data file:

```
date, val1, val2, val3, val4
94-100-12:34:56,   3.2,   1,   5,  3.1e8
94-100-12:44:57,   3.2,   0,   5,  3.1e8
94-100-12:54:58,   3.2,   1,   5,  3.1e8
94-100-13:04:59,   3.2,   1,   5,  3.1e8
94-101-12:24:60,   3.2,   1,   5,  3.1e8
94-101-12:34:61,   3.2,   1,   5,  3.1e8
94-101-12:44:62,   2.9,   1,   5,  3.1e8
94-101-12:54:63,   2.0,   1,   5,  3.1e8
94-101-13:04:64,   2.5,   1,   5,  3.1e8
94-101-13:14:65,   3.0,   1,   5,  3.1e8
```

And this thresholds file:

```
3.0     val1
1.0     val2
7.2     val3
1e4     val4
```

Execute dsp_timeon and get:

```
date, val1, val2, val3, val4
94-100-00:00:00, 30:03, 20:02, 0, 30:03
94-101-00:00:00, 20:02, 50:05, 0, 50:05
Totals:          50:05, 01:10:07, 0, 01:20:08
```

New data may be appended to this file. The '`Totals`' line will be ignored when the file is read on subsequent runs, the existing daily totals will be preserved or augmented, and new grand totals will be calculated.

Note that any non-numeric values in the thresholds file will be ignored along with any numbers after the first in each row. If no thresholds file is specified, default thresholds of 0.5 are used for each column.

See section "Plot data files" in *ITOS Plotting Users' Guide* for the formatting rules for the data values in the data and threshold files.

TIMEOUT

`TIMEOUT number`

This directive may optionally have a preceding slash, as in `/TIMEOUT 20`.

This directive is only allowed when commanding is enabled (see [ENABLE], page 24).

The `TIMEOUT` directive sets a time interval over which the ground expects to receive a status report (CLCW) from the spacecraft about a command that has been transmitted.

UNLOCK

`UNLOCK`

This directive may optionally have a preceding slash, as in `/UNLOCK`

This directive is only allowed when commanding is enabled (see [ENABLE], page 24).

The `UNLOCK` directive sends a control command to the spacecraft which unlocks the spacecraft so it can accept `CMD` commands.

Note that control commands are transmitted immediately, even when in two step command mode. See [MODE], page 48.

See [RESET], page 72.

VC

`VC number`

This directive may optionally have a preceding slash, as in `/VC 1`

The `VC` directive specifies the virtual channel over which subsequent spacecraft commands shall be transmitted.

VERIFY

```
VERIFY [loadfile] [, dumpfile] [> reportfile]
```

`loadfile` and `dumpfile` are strings (either quoted strings or string expressions), and `reportfile` is string or parenthesized string expression.

`loadfile` is the name of an image load file. If `loadfile` contains a `/` it is the pathname to the file; otherwise `loadfile` names the file in the default load file directory (global mnemonic *gbl_imgloaddir*). If omitted, the value of most recently loaded file (global mnemonic *gbl_loadfile*) is used.

`dumpfile` is the name of an image dump file. If `dumpfile` contains a `/` it is the pathname to the file; otherwise `dumpfile` names the file in the default dump file directory (global mnemonic *gbl_imgdumpdir*). If omitted, the value of most recently dumped file (global mnemonic *gbl_dumpfile*) is used.

`reportfile` is the name of a report file. If `reportfile` contains a `/` it is the pathname to the file; otherwise `reportfile` names the file in the default report file directory (global mnemonic *gbl_imgreportdir*). If omitted, the report will be written to a file with the same name as the dumpfile but with the extension ".RPT". The global mnemonic *gbl_reportfile* contains the name of the most recently created report file.

The `VERIFY` directive is used for image verification. A byte-by-byte comparison of the data in each file is performed and the differences are shown in a comparison report which is located in *gbl_reportfile*. The global mnemonic *gbl_miscmp* contains the number of miscompared bytes. The global mnemonic *gbl_cmpdone* is set to the value of 1 when the compare is done.

If desired, `VERIFY` can be used to compare two load files or two dump files.

WAIT

```
WAIT [howlong] [UNTIL ( condition )] [ options ...]
```

howlong – A float value indicating the maximum amount of time, in seconds, to wait. If not specified or zero, the wait is indefinite. If negative and no condition is specified then the wait is ignored. If negative and condition is specified then wait will not end until the condition is true. For historical compatibility, may optionally be followed by a comma and another float value; the second value is ignored. This option is depricated and will no longer be supported.

condition – a condition is a STOL expression containing telemetry mnemonics that when evaluates TRUE will cause the wait to end. If no mnemonics are in the condition expression then the expression will be reevaluated once a second until true. If howlong is specified then the wait will end after that time period expires even though the condition is still false.

options – space seperated list of options in any order to include:

`STATUS=`*variable*
: Specifies a local or global variable or integer mnemonic to receive the status from the wait *condition*. States are 1 == SUCCESS, 0 == TIMEOUT. If SUCCESS then the *condition* was met. If TIMEOUT then *howlong* time has elapsed and the *condition* has not been met.

`VERBOSE`
: This flag causes values of for all mnemonics in *condition* to be displayed as they change.

The WAIT directive is used to temporarily pause a proc. Some examples of how the WAIT directive is used:

```
; wait three seconds
wait 3


; send a command and wait for end-item verification
local orgtemp
orgtemp = heatertemp ; heatertemp is a database mnemonic
/turnonheater
wait until (heatertemp .gt. (orgtemp + 2)) ; if heater was turned on,
                                           ; heatertemp should rise


; above, with a five second timeout
local orgtemp, success
orgtemp = heatertemp
/turnonheater
wait 5 until (heatertemp .gt. (orgtemp + 2)) status=success
```

```
if (success) then
   ; actually got end-item verification
   ...
else
   ; timed out - did not get end-item verification
   ...
endif


; wait until a specific time (works because an absolute date minus an
; absolute date is a relative time, and a relative time can be converted
; to float; the expression reduces to ``wait howlong'')
local x
x = 00-199-14:23:00
wait x - p@gbl_gmtoff
```

See [GO], page 34.

WRITE

```
WRITE ( unit [ options ...] ) value [, value ...]
```

unit – int constant, variable, or expression identifying the unit to be read.

value – identifies a local variable, global variable, or string.

Options are space seperated and may be specified in any order and include:

`NOEOL` Specifies that no end of line character ("\n") is output after all values are output. Default is to send an EOL.

`STATUS=`*variable*
Specifies a local or global variable or integer mnemonic to receive the status from the write operation. States are 1 == SUCCESS, 0 == CONNECTION CLOSED, -1 == ERROR.

It is up to the user to test the return *variable* before attempting another WRITE.

If STATUS is specified the proc will continue regardless of the status value. Otherwise proc will stop with any condition other than SUCCESS.

Writes ascii data to a device or file. Embedded "\" sequences are converted to actual hex byte codes for the following:

`\a ==> 0x07` *(bell)*, `\b ==> 0x08` *(backspace)*,
`\f ==> 0x0C` *(formfeed)*, `\g ==> 0x07` *(bell)*,
`\n ==> 0x0A` *(newline)*, `\r ==> 0x0D` *(carriage return)*,
`\t ==> 0x09` *(horz tab)*, `\v ==> 0x0B` *(vert tab)*,
`\\ ==> 0x5C` *(backslash)*, `\0 ==> 0x00` *(null)*.

Some examples:

```
OPEN(1) "/dev/ttyb"
WRITE(1) "5/8 = ", 5/8, "; 5.0/8.0 = ", 5.0/8.0
```

writes `5/8 = 0; 5.0/8.0 = 0.625` to serial port b;

```
LOCAL evtlog
LOCAL fstatus

evtlog = 5
OPEN(evtlog) (gbl_evtfifo), write
WRITE(evtlog status=fstatus) "hi mom"
if (fstatus .LT. 1) then
    ask "WRITE to event log failed!"
    return halted
endif
```

writes `hi mom` as a NULL_EVENT to the event log.

```
OPEN (1) "testhost:7000"
WRITE (1 NOEOL) "This is a test\r"
```

writes "`This is a test\r`" to testhost computer on port 7000. The string has a carriage return at the end and no newline.

See [OPEN], page 49 for how to open a unit. See [READ], page 69 for how to read from an open unit.

ZERO

The *ZERO* directive clears ITOS telemetry counters.

Syntax

`ZERO`

Clear all active ITOS telemetry counters.

Discussion

The `zero` directive clears all active ITOS telemetry counters. This includes all counters updated by `frame_sorter` and `tlmClient` processes. Note that only counter being updated by running processes are reset. If the telemetry subsystem is disabled or is not acquiring telemetry, the counters will not be changed. Counters are cleared automatically when telemetry processing is initiated.